程序员的掌中宝

易携带、双目录、示例多
学、练、用一本就GO！



Java 全能速查宝典

明日科技 李永强 李钟尉 王国辉 编著

1172个条目，1025个示例，Java编程随时查
电话、QQ、TQ、论坛，四位立体式服务

人民邮电出版社
POSTS & TELECOM PRESS

程序员的掌中宝

易携带、双目录、示例多
学、练、用一本就GO！

Java 全能速查宝典

明日科技 李永强 李钟尉 王国辉 编著

人民邮电出版社
北京

PDF电子书基地 http://dayo1982.400gb.com

图书在版编目（CIP）数据

Java全能速查宝典 / 李永强，李钟尉，王国辉编著
-- 北京 : 人民邮电出版社，2009.11
ISBN 978-7-115-21487-4

Ⅰ. ①J… Ⅱ. ①李… ②李… ③王… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2009)第183658号

内 容 提 要

Java 编程的最基本要素是方法、属性和事件，掌握这些要素，就掌握了解决实际问题的一般方法。本书从 Java SE 及 Java EE 两方面入手，精选了 JDK 自身 600 多个常用的方法、属性，同时还对 Jsp、Servlet 技术及 Java EE 流行框架的使用进行全面的解析，每一个知识点都配有具体的示例，便于读者理解。

本书所讲的知识点按照功能和字母顺序进行排序，读者既可以按照功能顺序查找，也可以按照字母顺序学习。

本书不仅适合 Java 程序设计初学者，也可作为中、高级程序开发人员的参考手册。

Java 全能速查宝典

◆ 编　　著　明日科技　李永强　李钟尉　王国辉
责任编辑　屈艳莲

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：880×1230　1/32
印张：26.625
字数：882 千字　　2009 年 11 月第 1 版
印数：1－3 500 册　　2009 年 11 月北京第 1 次印刷

ISBN 978-7-115-21487-4

定价：59.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

在众多编程语言之中，Java 语言是第一个纯粹面向对象的编程语言，它以这一纯朴的本质，吸引了越来越多的程序员。当然 Java 的优点不仅仅于此，在编程领域之中，其先进的编程思想、经典的设计模式，都体现得淋漓尽致，目前它已成为最为流行的编程语言之一。

随着 JDK 版本的不断升级，Java 已经历了无数次的飞跃，它所涵盖的内容更为丰富。而对于 Java 程序开发人员来说，拥有一本速查手册工具书极其必要，因此笔者编写了本书，希望真正帮助 Java 程序开发人员解决易查、易学、易用的问题，并为您成为程序开发高手打下坚实的基础。

本书特点

示例丰富

通过实例学习是最好的学习方式，本书每个知识点都对应有相关示例。便于读者快速掌握知识点。

贴近实际

本书对一些重点的属性、方法、类及技术进行讲解时，都提供了典型示例，以便让读者掌握其实际应用。

查找方便

本书可按两种方式进行查询，一是按照函数、语句、控件的功能查询，二是按照首字母顺序查询，方便用户的使用和查找。

本书内容

本书共分为 22 章，包括 JDK 常用类库、Jsp 应用及 Java EE 轻量级框架等丰富技术，其内容丰富实用，涉及 Java 中常用类的方法及属性、Jsp 编程、JSTL 标签库、Struts 框架、Hibernate 框架、Spring 框架等各方面技术内容。

读者对象

- 初学编程的自学者
- 大中专院校的老师和学生
- 初中级程序开发人员
- 参加实习的菜鸟程序员
- 编程爱好者
- 相关培训机构的老师和学员
- 程序测试及维护人员

PDF电子书基地 http://dayo1982.400gb.com

本书服务

如果您在使用本书时遇到什么困难或疑惑，可以联系我们，我们将在 5 个工作日内给您提供解答。我们的服务方式如下：

服务网站：www.mingribook.com

服务 QQ：100310286　100310063

服务信箱：mingrisoft@mingrisoft.com

服务电话：0431-84978981/84978982

致读者

本书由 Java 程序开发团队策划并组织编写，主要编写人员有李永强、李钟尉、王国辉等，在编写本书的过程中，我们以科学、严谨的态度，力求精益求精，但错误、疏漏之处在所难免，敬请广大读者批评指正。

编者

2009 年 8 月

目　录

第 1 章　字符串　1

1.1　charAt 方法——提取指定字符　1

1.2　codePointAt 方法——提取索引字符代码点　1

1.3　codePointBefore 方法——获取索引前一个字符的代码点　2

1.4　codePointCount 方法——获取指定范围文本代码点数　2

1.5　compareTo 方法——比较两个字符串　3

1.6　compareToIgnoreCase 方法——忽略大小写比较字符串　4

1.7　concat 方法——字符串结尾连接　5

1.8　contains 方法——判断是否包含指定字符　5

1.9　copyValueOf 方法——字符数组生成字符串　6

1.10　endsWith 方法——判断后缀字符串　7

1.11　equals 方法——判断字符串相等　7

1.12　equalsIgnoreCase 方法——忽略大小写判断字符串相等　8

1.13　format 方法——格式化字符串　8

1.14　getBytes 方法——获取字符串的字节数组　11

1.15　getChars 方法——获取字符数组　12

1.16　hashCode 方法——生成字符串哈希表　13

1.17　indexOf 方法——获取字符第一个索引　13

1.18　intern 方法——获取规范化字符串　15

1.19　isEmpty 方法——判断字符串是否为空　16

1.20　lastIndexOf 方法——获取字符最后的索引　16

1.21　length 方法——获取字符串长度　18

1.22　matches 方法——匹配正则表达式　18

1.23　offsetByCodePoints 方法——获取索引偏移后指定代码点的索引　19

1.24　regionMatches 方法——测试两个字符串区域是否相等　19

1.25　replace 方法——替换字符序列　21

1.26　replaceAll 方法——正则表达式匹配替换所有字符串　22

1.27　replaceFirst 方法——正则表达式替换第一个匹配的字符串　23

PDF电子书基地 http://dayo1982.400gb.com

1.28 split 方法——字符串分割成数组 23
1.29 startsWith 方法——判断前缀字符串 25
1.30 subSequence 方法——获取子字符串序列 26
1.31 substring 方法——获取子字符串 26
1.32 toCharArray 方法——字符串变字符数组 27
1.33 toLowerCase 方法——转换成小写字符串 27
1.34 toUpperCase 方法——转换成大写字符串 28
1.35 trim 方法——截取空白 29
1.36 valueOf 方法——基本数据类型转成字符串 29

第 2 章 包装类 34
2.1 Integer 类——整数类 34
2.1.1 构造方法——Integer 类的构造方法 34
2.1.2 常量——Integer 类的常量 34
2.1.3 bitCount 方法——获取二进制补码中 1 的数量 34
2.1.4 byteValue 方法——获取 byte 类型的值 35
2.1.5 compareTo 方法——比较整数 35
2.1.6 decode 方法——字符串解码为 int 类型 35
2.1.7 doubleValue 方法——返回 double 数值 36
2.1.8 equals 方法——判断整数对象相等 36
2.1.9 floatValue 方法——获取 float 数值 37
2.1.10 getInteger 方法——获取整数的系统属性值 37
2.1.11 hashCode 方法——生成整数的哈希码 39
2.1.12 highestOneBit 方法——获取整数二进制最高位 1 的索引 39
2.1.13 intValue()方法——获取 int 值 40
2.1.14 longValue 方法——获取 long 值 40
2.1.15 lowestOneBit 方法——获取整数二进制最低位 1 的索引 41
2.1.16 parseInt 方法——将字符串解析为 int 值 41
2.1.17 reverse 方法——反转整数二进制补码的位顺序 43
2.1.18 reverseBytes 方法——反转整数字节的顺序 44
2.1.19 shortValue 方法——获取 short 值 44

2.1.20 signum 方法——获取整数符号 44
2.1.21 toBinaryString 方法——生成整数的二进制字符串 45
2.1.22 toHexString 方法——生成整数的十六进制字符串 45
2.1.23 toOctalString 方法——生成整数的八进制字符串 46
2.1.24 toString 方法——生成整数的十进制字符串 47
2.1.25 valueOf 方法——创建 Integer 对象 49
2.2 Long 类——长整型类 50
2.2.1 构造方法——Long 类的构造方法 51
2.2.2 常量——Long 类的常量 51
2.2.3 bitCount 方法——获取二进制补码中 1 的数量 51
2.2.4 byteValue 方法——获取 byte 值 51
2.2.5 compareTo 方法——比较长整数 52
2.2.6 decode 方法——字符串解码为 long 类型 52
2.2.7 doubleValue 方法——返回 double 数值 53
2.2.8 equals 方法——判断长整数对象相等 53
2.2.9 floatValue 方法——获取 float 数值 54
2.2.10 getLong 方法——获取长整数的系统属性值 54
2.2.11 hashCode 方法——生成长整数的哈希码 56
2.2.12 highestOneBit 方法——获取长整数二进制最高位 1 的索引 56
2.2.13 intValue()方法——获取 int 值 57
2.2.14 longValue 方法——获取 long 值 57
2.2.15 lowestOneBit 方法——获取长整数二进制最低位 1 的索引 57
2.2.16 parseLong 方法——将字符串解析为 long 值 58
2.2.17 reverse 方法——反转长整数二进制补码的位顺序 60
2.2.18 reverseBytes 方法——反转长整数字节的顺序 60
2.2.19 shortValue 方法——获取 short 值 61
2.2.20 signum 方法——获取长整数符号 61
2.2.21 toBinaryString 方法——生成长整数的二进制字符串 61
2.2.22 toHexString 方法——生成长整数的十六进制字符串 62
2.2.23 toOctalString 方法——生成长整数的八进制字符串 62
2.2.24 toString 方法——生成长整数的十进制字符串 63
2.2.25 valueOf 方法——创建 Long 对象 65

PDF电子书基地 http://dayo1982.400gb.com

2.3 Short 类——短整型类 67
2.3.1 构造方法——Short 类的构造方法 67
2.3.2 常量——Short 类的常量 67
2.3.3 compareTo 方法——比较短整数 67
2.3.4 decode 方法——字符串解码为 short 类型 68
2.3.5 doubleValue 方法——返回 double 数值 68
2.3.6 equals 方法——判断短整数对象相等 68
2.3.7 floatValue 方法——获取 float 数值 69
2.3.8 hashCode 方法——生成短整数的哈希码 69
2.3.9 intValue()方法——获取 int 值 70
2.3.10 longValue 方法——获取 long 值 70
2.3.11 parseShort 方法——将字符串解析为 short 值 70
2.3.12 reverseBytes 方法——反转短整数字节的顺序 72
2.3.13 shortValue 方法——获取 short 值 72
2.3.14 toString 方法——生成短整数的十进制字符串 73
2.3.15 valueOf 方法——创建 Short 对象 73
2.4 Boolean 类——布尔类 75
2.4.1 构造方法——Boolean 类的构造方法 75
2.4.2 常量——Boolean 类的常量 75
2.4.3 booleanValue 方法——获取 boolean 值 76
2.4.4 compareTo 方法——比较布尔值 76
2.4.5 equals 方法——判断相等 77
2.4.6 getBoolean 方法——获取布尔类型的系统属性值 77
2.4.7 hashCode 方法——生成布尔对象的哈希码 78
2.4.8 parseBoolean 方法——将字符串解析成 boolean 值 78
2.4.9 toString 方法——生成布尔值的字符串 78
2.4.10 valueOf 方法——创建布尔对象 79
2.5 Byte 类——字节对象 80
2.5.1 构造方法——Byte 类的构造方法 80
2.5.2 常量——Byte 类的常量 80
2.5.3 compareTo 方法——比较字节对象 80
2.5.4 decode 方法——将字符串解码为 Byte 值 81
2.5.5 doubleValue 方法——获取 double 值 82

2.5.6 equals 方法——判断字节相等 82
2.5.7 floatValue 方法——获取 float 值 83
2.5.8 hashCode 方法——生成字节对象的哈希码 83
2.5.9 intValue 方法——获取 int 值 83
2.5.10 longValue 方法——获取 long 值 83
2.5.11 parseByte 方法——将字符串解析为 byte 值 84
2.5.12 shortValue 方法——获取 short 值 85
2.5.13 toString 方法——生成字节值的十进制字符串 85
2.5.14 valueOf 方法——创建 Byte 对象 86
2.6 Character 类——字符类 88
2.6.1 构造方法——Character 类的构造方法 88
2.6.2 常量——Character 类的常量 88
2.6.3 charCount 方法——计算指定字符代码点的数量 89
2.6.4 charValue 方法——获取 char 值 89
2.6.5 codePointAt 方法——获取字符数组元素的代码点 90
2.6.6 codePointBefore 方法——获取字符数组索引前一个元素的代码点 91
2.6.7 codePointCount 方法——返回字符数组的子数组中代码点的数量 93
2.6.8 compareTo 方法——比较字符对象 94
2.6.9 equals 方法——判断字符对象相等 95
2.6.10 getNumericValue 方法——返回字符表示的 int 值 95
2.6.11 getType 方法——返回一个指示字符的常规类别的值 97
2.6.12 hashCode 方法——生成字符对象的哈希码 97
2.6.13 isDefined 方法——判断是否为 Unicode 字符 98
2.6.14 isDigit 方法——判断是否为数字字符 98
2.6.15 isLetter 方法——判断是否为字母字符 99
2.6.16 isLowerCase 方法——判断是否为小写字符 100
2.6.17 isUpperCase 方法——判断是否为大写字符 100
2.6.18 toLowerCase 方法——转换为小写字符 101
2.6.19 toUpperCase 方法——转换为大写字符 101
2.7 Double——双精度数字类 102
2.7.1 构造方法——Double 类的构造方法 102

PDF电子书基地 http://dayo1982.400gb.com

2.7.2 常量——Double 类的常量 102
2.7.3 byteValue 方法——获取 byte 值 102
2.7.4 compare 方法——比较双精度数字对象 103
2.7.5 compareTo 方法——比较两个 Double 对象 103
2.7.6 intValue 方法——将此 Double 值以 int 形式返回 104
2.7.7 doubleToLongBits 方法——返回指定浮点值的表示形式 104
2.7.8 doubleToRawLongBits 方法——保留 NaN 值返回指定浮点值的表示形式 105
2.7.9 doubleValue 方法——获取 double 值 105
2.7.10 equals 方法——判断 Double 对象是否相等 106
2.7.11 floatValue 方法——获取 float 值 107
2.7.12 hashCode 方法——生成 Double 对象的哈希码 107
2.7.13 isInfinite 方法——判断 Double 值的大小是否是无穷大 107
2.7.14 isNaN 方法——判断 Double 值是否是一个非数字值 108
2.7.15 longBitsToDouble 方法——返回给定位表示形式的 double 值 109
2.7.16 longValue 方法——获取 long 值 110
2.7.17 parseDouble 方法——将字符串解析为 double 值 110
2.7.18 shortValue 方法——获取 short 值 110
2.7.19 toHexString 方法——生成双精度数字的十六进制字符串 111
2.7.20 toString 方法——生成双精度数字的十进制字符串 112
2.7.21 valueOf 方法——创建 Double 对象 112
2.8 Float——浮点类 113
2.8.1 构造方法——Float 类的构造方法 113
2.8.2 常量——Float 类的常量 114
2.8.3 byteValue 方法——获取 byte 值 114
2.8.4 compare 方法——比较 Float 对象 114
2.8.5 compareTo 方法——比较两个 Float 对象所表示的数值 115
2.8.6 doubleValue 方法——获取 double 值 115
2.8.7 equals 方法——判断 Double 对象相等 115
2.8.8 floatToIntBits 方法——返回浮点值的表示形式 116

2.8.9 floatToRawIntBits 方法——保留非数字值返回指定浮点值的表示形式 117
2.8.10 floatValue 方法——获取 float 值 118
2.8.11 hashCode 方法——返回 Float 对象的哈希码 118
2.8.12 intBitsToFloat 方法——返回指定位表示形式的 float 值 118
2.8.13 intValue 方法——获取 int 值 119
2.8.14 isInfinite 方法——判断 float 值的大小是否是无穷大 120
2.8.15 isNaN 方法——判断 Float 值是否是一个非数字值 120
2.8.16 longValue 方法——获取 long 值 121
2.8.17 parseFloat 方法——将字符串解析成 float 值 121
2.8.18 shortValue 方法——获取 short 值 122
2.8.19 toHexString 方法——生成浮点数的十六进制字符串 122
2.8.20 toString 方法——生成浮点数的十进制字符串 123
2.8.21 valueOf 方法——创建浮点数对象 124

第 3 章 输入输出 125
3.1 BufferedInputStream 类——缓冲字节输入流 125
3.1.1 构造方法——BufferedInputStream 类的构造方法 125
3.1.2 available 方法——返回可以读取的估计字节数 125
3.1.3 close 方法——关闭此输入流并释放所有关联的系统资源 126
3.1.4 mark 方法——记录当前指针所在的位置 126
3.1.5 markSupported 方法——测试是否支持 mark()方法和 reset()方法 127
3.1.6 read 方法——从输入流中读取数据 128
3.1.7 reset 方法——重新定位输入流 129
3.1.8 skip 方法——跳过并丢弃指定字节数量的数据 129
3.2 BufferedOutputStream 类——缓冲字节输出流 130
3.2.1 构造方法——BufferedOutputStream 类的构造方法 130
3.2.2 flush 方法——刷新此缓冲输出流 131
3.2.3 write 方法——向输出流中写数据 131
3.3 BufferedReader 类——缓冲字符输入流 133
3.3.1 构造方法——BufferedReader 类的构造方法 133

PDF电子书基地 http://dayo1982.400gb.com

3.3.2 close 方法——关闭流并释放与之关联的所有资源 133
3.3.3 mark 方法——记录当前读指针所在的位置 134
3.3.4 markSupported 方法——判断指定流是否支持 mark()操作 134
3.3.5 read 方法——从缓冲输入流中读取字符 135
3.3.6 readLine 方法——读取一个文本行 136
3.3.7 ready 方法——判断此流是否已准备好被读取 137
3.3.8 reset 方法——重定位指针 137
3.3.9 skip 方法——跳过指定数量的字符 138
3.4 BufferedWriter 类——缓冲字符输出流 138
3.4.1 构造方法——BufferedWriter 类的构造方法 139
3.4.2 close 方法——关闭流 139
3.4.3 flush 方法——刷新缓冲区 139
3.4.4 newLine 方法——写入一个行分隔符 140
3.4.5 write 方法——向缓冲区中写数据 141
3.5 DataInputStream 类——数据输入流 142
3.5.1 read 方法——读取一定数量的字节 143
3.5.2 readByte 方法——读取此操作需要的字节 145
3.5.3 readBoolean 方法——读取一个布尔值 145
3.5.4 readchar 方法——读取此操作需要的字符 145
3.5.5 readDouble 方法——读取一个 double 值 146
3.5.6 readFloat 方法——读取一个 float 值 146
3.5.7 readFully 方法——读取一些字节 147
3.5.8 readInt 方法——读取一个 int 值 148
3.5.9 readLong 方法——读取一个 long 值 149
3.5.10 readShort 方法——读取一个 short 值 149
3.5.11 readUnsignedByte 方法——读取无符号的 byte 值 149
3.5.12 readUnsignedShort 方法——读取无符号的 short 值 150
3.5.13 readUTF 方法——读取 UTF-8 编码的字符串 150
3.5.14 skipBytes 方法——在输入流中跳过数据的 *n* 个字节 152
3.6 DataOutputStream 类——数据输出流 153
3.6.1 flush 方法——清空此数据输出流 153
3.6.2 size 方法——返回计数器 written 的当前值 154

3.6.3 write 方法（流输出方法） 155
3.6.4 writeBoolean 方法——将 boolean 值写入输出流 156
3.6.5 writeByte 方法——将 byte 值写入输出流 156
3.6.6 writeBytes 方法——将字符串中每一个字符的低字节的内容写入输出流 157
3.6.7 writeChar 方法——将 char 值写入输出流 157
3.6.8 writechars 方法——将字符串每个 char 值写入输出流 158
3.6.9 writeDouble 方法——将 double 值写入输出流 158
3.6.10 writeFloat 方法——将 float 值写入输出流 159
3.6.11 writeLong 方法——将 long 值写入输出流 160
3.6.12 writeShort 方法——将 short 值写入输出流 160
3.6.13 writeUTF()方法——将 UTF-8 编码字符写入输出流 161
3.7 File 类——文件类 161
3.7.1 构造方法——File 类的构造方法 161
3.7.2 常量——File 类的常量 162
3.7.3 canExecute 方法——判读文件是否可执行 162
3.7.4 canRead 方法——判断文件是否可读 162
3.7.5 canWrite 方法——判断文件是否可写 163
3.7.6 compareTo 方法——比较文件对象 163
3.7.7 createNewFile 方法——创建新的文件 164
3.7.8 createTempFile 方法——创建临时文件 164
3.7.9 delete 方法——删除文件 167
3.7.10 deleteOnExit 方法——在虚拟机终止时，请求删除文件 167
3.7.11 equals 方法——判断 File 对象是否相等 168
3.7.12 exists 方法——判断文件是否存在 168
3.7.13 getAbsoluteFile 方法——获取绝对路径的文件对象 169
3.7.14 getAbsolutePath 方法——获取绝对路径的字符串对象 169
3.7.15 getCanonicalFile 方法——返回此抽象路径名的规范形式 170
3.7.16 getCanonicalPath 方法——返回此抽象路径名的规范路径名字符串 170
3.7.17 getFreeSpace 方法——获取剩余空间 171

PDF电子书基地 http://dayo1982.400gb.com

3.7.18 getName 方法——获取文件名称 172
3.7.19 getParent 方法——获取上一级路径的字符串 172
3.7.20 getParentFile 方法——获取上一级路径的 File 对象 173
3.7.21 getPath 方法——获取路径字符串 173
3.7.22 getTotalSpace 方法——获取总空间大小 173
3.7.23 getUsableSpace 方法——精确获取可写入空间大小 174
3.7.24 hashCode 方法——生成 File 对象的哈希码 174
3.7.25 isAbsolute 方法——测试此路径名是否为绝对路径名 175
3.7.26 isDirectory 方法——判断 File 对象是否为目录 175
3.7.27 isFile 方法——判断 File 对象是否为文件 176
3.7.28 isHidden 方法——判断 File 对象是否设置隐藏属性 176
3.7.29 lastModified 方法——获取最后修改时间 177
3.7.30 length 方法——获取文件长度 177
3.7.31 list 方法——获取目录列表的字符串数组 178
3.7.32 listFiles 方法——获取目录列表的 File 数组 179
3.7.33 listRoots 方法——获取根目录的 File 数组 181
3.7.34 mkdir 方法——创建目录 182
3.7.35 mkdirs 方法——创建完整路径的所有目录 183
3.7.36 renameTo 方法——重命名文件或文件夹 183
3.7.37 setExecutable 方法——设置可执行属性 184
3.7.38 setLastModified 方法——设置最后修改时间 185
3.7.39 setReadable 方法——设置可读属性 185
3.7.40 setReadOnly 方法——设置只读属性 187
3.7.41 setWritable 方法——设置可写属性 187
3.7.42 toString 方法——File 的字符串格式 188
3.7.43 toURI 方法——获取 File 的 URI 189
3.8 FileInputStream 类——文件字节输入流 189
3.8.1 构造方法——FileInputStream 类的构造方法 189
3.8.2 available 方法——获取可读的有效字节数 189
3.8.3 close 方法——关闭文件输入流 190
3.8.4 finalize 方法——在销毁改对象时关闭输入流 191
3.8.5 getChannel 方法——返回与此文件输入流有关的惟一 FileChannel 对象 191

3.8.6 getFD 方法——获取 File 对象连接的 FileDescriptor 对象 191
3.8.7 read 方法——从文件字节流读取数组 192
3.8.8 skip 方法——跳过并丢弃 *n* 个字节 193
3.9 FileOutputStream 类——文件字节输出流 194
3.9.1 构造方法——FileOutputStream 类的构造方法 194
3.9.2 close 方法——关闭文件输出流并释放资源 195
3.9.3 finalize 方法——流销毁时关闭输出流 195
3.9.4 getChannel 方法——获取输出流的 FileChannel 对象 195
3.9.5 getFD 方法——获取 File 对象连接的 FileDescriptor 对象 196
3.9.6 write 方法——向文件输出流写数据 196
3.10 FileReader 类——文件的字符输入流 199
3.10.1 构造方法——FileReader 类的构造方法 199
3.10.2 close 方法——关闭字符输入流 200
3.10.3 read 方法——读取字符数据 200
3.11 FileWriter 类——文件的字符输出流 202
3.11.1 构造方法——FileWriter 类的构造方法 202
3.11.2 close 方法——关闭字符输出流 202
3.11.3 flush 方法——刷新缓冲区 203
3.11.4 write 方法——向字符输出流写数据 203
3.12 InputStream 类——字节输入流 205
3.12.1 available 方法——获取有效的可读取的字节数 205
3.12.2 close 方法——关闭输入流 206
3.12.3 mark 方法——在此输入流中标记当前的位置 206
3.12.4 markSupported 方法——判断是否支持 mark 和 reset 方法 207
3.12.5 read 方法——读取字节输入流中的数据 207
3.12.6 reset 方法——重新定位到最后一次 mark()方法时的位置 209
3.12.7 skip 方法——跳过并丢弃 *n* 个字节 209
3.13 OutputStream 类——字节输出流 210
3.13.1 close 方法——关闭字节输出流 210

PDF电子书基地 http://dayo1982.400gb.com

3.13.2 flush 方法——刷新缓冲区 211
3.13.3 write 方法——向输出流写数据 211
3.14 ObjectOutputStream 类——对象输出流 213
3.14.1 构造方法——ObjectOutputStream 类的构造方法 213
3.14.2 close 方法——关闭对象输出流 213
3.14.3 flush 方法——刷新缓冲区 214
3.14.4 writeObject 方法——向输出流写入对象 214
3.14.5 writeBytes 方法——向输出流写入字符串 215
3.14.6 write 方法——向输出流写入 byte 值 215
3.14.7 writeChar 方法——向输出流写入 char 值 216
3.14.8 writeChars 方法——向输出流写入 String 值 217
3.14.9 writeDouble 方法——向输出流写入 double 值 217
3.14.10 writeFields 方法——将已缓冲的字段写入流中 217
3.14.11 writeFloat 方法——向输出流写入 float 值 218
3.14.12 writeInt 方法——向输出流写入 int 值 218
3.14.13 writeLong 方法——向输出流写入 long 值 219
3.14.14 writeShort 方法——向输出流写入 short 值 219
3.14.15 writeUTF 方法——向输出流写入 float 值 219
3.15 ObjectInputStream 类——对象输入流 220
3.15.1 构造方法——ObjectInputStream 类的构造方法 220
3.15.2 readObject 方法——从输入流读取对象 220
3.15.3 readInt 方法——从输入流读取 int 值 221
3.15.4 readFloat 方法——从输入流读取 float 值 221
3.15.5 readchar 方法——从输入流读取 char 值 222
3.16 RandomAccessFile 类——随机文件访问类 222
3.16.1 构造方法——RandomAccessFile 类的构造方法 222
3.16.2 close 方法——关闭数据流 222
3.16.3 getChannel 方法——返回此文件的 FileChannel 对象 223
3.16.4 getFD 方法——返回此流的不透明文件描述符对象 223
3.16.5 getFilePointer 方法——返回文件中的当前偏移量 224
3.16.6 length 方法——返回此文件的长度 224
3.16.7 read 方法——读取一个数据字节 225
3.16.8 readBoolean 方法——从文件读取一个 boolean 值 227

3.16.9 readByte 方法——从文件读取一个 byte 值 228
3.16.10 readInt 方法——从文件读取一个 int 值 228
3.16.11 readChar 方法——从文件读取一个 char 值 229
3.16.12 readDouble 方法——从文件读取一个 double 值 229
3.16.13 readFloat 方法——从文件读取一个 float 值 230
3.16.14 readFully 方法——从文件读取一个字节数组 230
3.16.15 readLine 方法——从此文件读取文本的下一行 232
3.16.16 readLong 方法——从文件读取一个 long 值 232
3.16.17 readShort 方法——从文件读取一个 short 值 233
3.16.18 readUnsignedByte 方法——从文件读取无符号 byte 值 234
3.16.19 readUnsignedShort 方法——从文件读取无符号 short 值 234
3.16.20 readUTF 方法——从此文件读取一个 UTF-8 字符串 235
3.16.21 setLength 方法——设置此文件的长度 235
3.16.22 seek 方法——将文件指针移至指定位置 236
3.16.23 skipBytes 方法——跳过并丢弃 *n* 个字节 236
3.16.24 write 方法——向此文件写入指定的字节 237
3.16.25 writeBoolean 方法——将 boolean 写入文件 239
3.16.26 writeByte 方法——将 byte 写入该文件 239
3.16.27 writeBytes 方法——按字节序列将该字符串写入该文件 240
3.16.28 writeChars 方法——写入作为字符数据的字符串 240
3.16.29 writeDouble 方法——写入 double 值 241
3.16.30 writeFloat 方法——向文件写入 float 值 241
3.16.31 writeInt 方法——向文件写入 int 值 242
3.16.32 writeLong 方法——向文件写入 long 值 242
3.16.33 writeShort 方法——向文件写入 short 值 243
3.16.34 writeUTF 方法——向文件写入 UTF-8 编码的字符串 243
3.17 Reader 类——读取字符流的抽象类 244
3.17.1 close 方法——关闭流 244
3.17.2 mark 方法——标记流中的当前位置 244
3.17.3 markSupported 方法——判断是否支持 mark()操作 245

PDF电子书基地 http://dayo1982.400gb.com

3.17.4 read 方法——读取流中的数据 245
3.17.5 ready 方法——判断是否准备读取此流 247
3.17.6 reset 方法——重置该流 248
3.17.7 skip 方法——跳过并丢弃 *n* 个字节 248
3.18 Scanner 类——简单文本扫描器 249
3.18.1 构造方法——Scanner 类的构造方法 249
3.18.2 close 方法——关闭扫描器 250
3.18.3 delimiter 方法——返回匹配分隔符的 Pattern 250
3.18.4 findInLine 方法——忽略分隔符查找下一个构造模式 250
3.18.5 findWithinHorizon 方法——试图查找下一个指定模式 251
3.18.6 hasNext 方法——扫描器的输入中是否有另一个标记 252
3.18.7 hasNextBigDecimal 方法——下一个标记是否默认基数中的一个 BigDecimal 254
3.18.8 hasNextBigInteger 方法——下一个标记是否默认基数中的一个 BigInteger 254
3.18.9 hasNextBoolean 方法——下一个标记是否为布尔值 255
3.18.10 hasNextByte 方法——下一个标记是否为 byte 值 256
3.18.11 hasNextDouble 方法——下一个标记是否为 double 值 256
3.18.12 hasNextFloat 方法——下一个标记是否为 float 值 257
3.18.13 hasNextInt 方法——下一个标记是否为 int 值 257
3.18.14 hasNextLine 方法——是否存在下一行 258
3.18.15 hasNextLong 方法——下一个标记是否为 long 值 259
3.18.16 hasNextShort 方法——下一个标记是否为 short 值 260
3.18.17 ioException 方法——最后抛出的 IOException 260
3.18.18 locale 方法——返回此扫描器的语言环境 261
3.18.19 match 方法——最后扫描操作的匹配结果 261
3.18.20 next 方法——读取下一个完整标记 262
3.18.21 nextBigDecimal 方法——读取下一个 BigDecimal 对象 263
3.18.22 nextBigInteger 方法——读取下一个 BigInteger 对象 264
3.18.23 nextBoolean 方法——读取下一个 boolean 值 265
3.18.24 nextByte 方法——读取下一个 byte 值 266

3.18.25 nextDouble 方法——读取下一个 double 值 267
3.18.26 nextFloat 方法——读取下一个 float 值 267
3.18.27 nextInt 方法——读取下一个 int 值 268
3.18.28 nextLine 方法——读取下一行文本 269
3.18.29 nextLong 方法——读取下一个 long 值 269
3.18.30 nextShort 方法——读取下一个 short 值 270
3.18.31 radix 方法——返回扫描器的默认基数 271
3.18.32 reset 方法——重置扫描器 271
3.18.33 skip 方法——忽略分隔符跳过匹配的输入信息 272
3.18.34 toString 方法——返回此 Scanner 的字符串表示形式 273
3.18.35 useDelimiter 方法——设置为指定分隔模式 273
3.18.36 useLocale 方法——设置为指定的语言环境 274
3.18.37 useRadix 方法——设置指定基数 274
3.19 Writer 类——写入字符流的抽象类 275
3.19.1 append 方法——添加指定字符 275
3.19.2 close 方法——关闭数据流 275
3.19.3 flush 方法——刷新流的缓冲 276
3.19.4 write 方法——向字符输出流中写数据 276

第 4 章 集合类 279
4.1 List 接口——有序集合类 279
4.1.1 add 方法——向列表中插入元素 279
4.1.2 addAll 方法——将指定 collection 添加到列表中 280
4.1.3 clear 方法——从列表中移除所有元素 282
4.1.4 contains 方法——判断列表中是否包含指定元素 283
4.1.5 containsAll 方法——判断列表中是否包含指定 collection 所有元素 283
4.1.6 equals 方法——比较指定的对象与列表是否相等 285
4.1.7 get 方法——获取列表指定位置的元素 285
4.1.8 set 方法——替换列表中指定位置的元素 286
4.1.9 hashCode 方法——返回列表的哈希码值 287
4.1.10 indexOf 方法——返回第一次出现指定元素的位置 287
4.1.11 lastIndexOf 方法——返回最后一次出现指定元素的位置 288

PDF电子书基地 http://dayo1982.400gb.com

4.1.12 isEmpty 方法——判断集合是否为空 289
4.1.13 iterator 方法——返回迭代器 290
4.1.14 listIterator 方法——返回列表迭代器 290
4.1.15 remove 方法——移除列表中的指定元素 292
4.1.16 removeAll 方法——从列表中移除指定 collection 包含的所有元素 293
4.1.17 retainAll 方法——保留指定 collection 包含的所有元素 294
4.1.18 size 方法——返回列表中元素的个数 294
4.1.19 subList 方法——获取列表中指定范围的子列表 295
4.1.20 toArray 方法——返回所有元素的数组 296
4.2 Map 接口——映射集合类 297
4.2.1 clear 方法——移除所有映射关系 297
4.2.2 containsKey 方法——判断是否包含指定的键名 298
4.2.3 containsValue 方法——判断是否包含指定的键值 298
4.2.4 equals 方法——判断是否与指定的对象相同 299
4.2.5 get 方法——返回指定键所映射的值 300
4.2.6 isEmpty 方法——判断是否为空 300
4.2.7 keySet 方法——获取 Map 集合的所有 key 301
4.2.8 put 方法——向指定索引位置添加对象 301
4.2.9 putAll 方法——追加另一个 Map 对象到当前 Map 集合 302
4.2.10 remove 方法——移除 Map 集合中指定键名的内容 303
4.2.11 size 方法——获取 Map 集合类的大小 304
4.2.12 values 方法——获取 Map 集合中所有键值对象 304
4.3 Set 接口——无重复元素集合类 305
4.3.1 add 方法——向 Set 集合中添加对象 305
4.3.2 addAll 方法——向 Set 集合添加另一个集合的所有内容 306
4.3.3 clear 方法——从 Set 集合中移除所有内容 307
4.3.4 contains 方法——判断 Set 集合是否包含指定对象 307
4.3.5 containsAll 方法——判断 Set 集合是否包含另一个集合中全部对象 308
4.3.6 equals 方法——比较指定对象与此 Set 是否相等 309
4.3.7 isEmpty 方法——判断 Set 集合是否为空 309

4.3.8 iterator 方法——获取 Set 集合的迭代器 310
4.3.9 remove 方法——移除 Set 集合中指定对象 311
4.3.10 removeAll 方法——移除另一个集合所包含的所有内容 311
4.3.11 retainAll 方法——保留另一个集合所包含的所有内容 312
4.3.12 size 方法——获取 Set 集合类的大小 313
4.3.13 toArray 方法——用 Set 集合的所有对象创建数组 313

第 5 章 数据库编程 315
5.1 DriverManager 类——驱动程序管理类 315
5.1.1 deregisterDriver 方法——删除一个驱动程序 315
5.1.2 getDriver 方法——选择一个适当的驱动程序 315
5.1.3 getDrivers 方法——获取全部已加载的驱动程序 316
5.1.4 getConnection 方法——获取数据库连接 316
5.1.5 getLoginTimeout 方法——获取连接数据库的等待时间 318
5.1.6 getLogWriter 方法——获取日志的打印输出流 319
5.1.7 println 方法——输出日志 319
5.1.8 registerDriver 方法——注册给定的驱动程序 319
5.1.9 setLoginTimeout 方法——设置连接数据库的等待时间 320
5.1.10 setLogWriter 方法——设置日志的打印输出流 320
5.2 Connection 接口——数据库连接接口 320
5.2.1 常量——Connection 接口的常量 321
5.2.2 clearWarnings 方法——清除警告信息 321
5.2.3 close 方法——关闭数据库连接对象 321
5.2.4 commit 方法——提交事务 321
5.2.5 createBlob 方法——构造二进制的对象 322
5.2.6 createClob 方法——构造字符型的对象 322
5.2.7 createNClob 方法——构造字节字符的对象 323
5.2.8 createSQLXML 方法——构造 SQLXML 对象 323
5.2.9 createStatement 方法——创建执行 SQL 语句的对象 324
5.2.10 getAutoCommit 方法——获取提交模式 326
5.2.11 getCatalog 方法——获取当前目录名称 326
5.2.12 getClientInfo 方法——获取客户端信息列表 326
5.2.13 getHoldability 方法——获取 ResultSet 对象的可保存性 327

PDF电子书基地 http://dayo1982.400gb.com

5.2.14 getMetaData 方法——获取数据库的元数据 328
5.2.15 getTransactionIsolation 方法——获取事务隔离级别 328
5.2.16 getTypeMap 方法——获取 Map 对象 329
5.2.17 getWarnings 方法——获取调用报告的第一个警告 329
5.2.18 isClosed 方法——判断 Connection 对象是否被关闭 330
5.2.19 isReadOnly 方法——判断 Connection 对象是否为只读模式 330
5.2.20 isValid 方法——判断连接是否有效 330
5.2.21 nativeSQL 方法——返回 SQL 语句的本机形式 331
5.2.22 prepareCall 方法——调用数据库存储过程 331
5.2.23 prepareStatement 方法——创建一个预处理语句 334
5.2.24 rollback 方法——事务回滚 335
5.2.25 setAutoCommit 方法——设置数据库连接的提交模式 335
5.2.26 setCatalog 方法——设置目录名称 336
5.2.27 setClientInfo 方法——设置客户端信息 336
5.2.28 setHoldability 方法——设置 ResultSet 对象的可保存性 337
5.2.29 setReadOnly 方法——设置连接只读模式 338
5.2.30 setSavepoint 方法——创建一个未命名的保存点 338
5.2.31 releaseSavepoint 方法——移除指定的 SavePoint 实例 339
5.2.32 setTransactionIsolation 方法——设置事务隔离级别 339
5.3 Statement 接口——执行 SQL 语句接口 340
5.3.1 常量——Statement 接口的常量 340
5.3.2 addBatch 方法——添加批处理语句 340
5.3.3 cancel 方法——中止 SQL 语句 341
5.3.4 clearBatch 方法——清除批处理语句 341
5.3.5 clearWarnings 方法——清除所有警告 341
5.3.6 execute 方法——执行 SQL 语句 342
5.3.7 executeBatch 方法——执行批处理语句 342
5.3.8 executeUpdate 方法——执行更新操作 343
5.3.9 executeQuery 方法——执行查询操作 344
5.3.10 getConnection 方法——获取数据库连接对象 344
5.3.11 getFetchDirection 方法——获取从数据库表获取行的方向 345

5.3.12 getFetchSize 方法——获取结果集的行数 345
5.3.13 getGeneratedKeys 方法——获取自动生成的键 346
5.3.14 getMaxFieldSize 方法——获取最大字节数 346
5.3.15 getMaxRows 方法——获取最大行数 347
5.3.16 getMoreResults 方法——移动到 Statement 对象的下一个结果 347
5.3.17 getQueryTimeout 方法——获取等待执行的秒数 348
5.3.18 getResultSet 方法——获取结果集 349
5.3.19 getResultSetConcurrency 方法——获取结果集并发性 349
5.3.20 getResultSetHoldability 方法——获取结果集可保存性 350
5.3.21 getResultSetType 方法——获取结果集类型 350
5.3.22 getUpdateCount 方法——获取更新的记录数 350
5.3.23 getWarnings 方法——获取调用报告的第一个警告 351
5.3.24 setMaxRows 方法——获取最大行数 351
5.3.25 getMaxRows 方法——获取最大行数限制值 352
5.3.26 close 方法——关闭 Statement 对象 352
5.3.27 isClosed 方法——判断 Statement 对象是否关闭 352
5.3.28 isPoolable 方法——判断对象是否可池化 353
5.3.29 setCursorName 方法——设置指针名称 353
5.3.30 setEscapeProcessing 方法——设置是否进行转义处理 354
5.3.31 setFetchDirection 方法——设置处理行的方向 354
5.3.32 setFetchSize 方法——设置结果集合的最大行数 355
5.3.33 setMaxFieldSize 方法——设置可返回的最大字节数 355
5.3.34 setMaxRows 方法——设置最大行数 356
5.3.35 setPoolable 方法——设置语句是否可池化 356
5.3.36 setQueryTimeout 方法——设置等待执行的秒数 357
5.4 PreparedStatement 接口——预编译 SQL 语句接口 357
5.4.1 clearParameters 方法——立即清除当前参数值 357
5.4.2 execute 方法——执行 SQL 语句 358
5.4.3 executeQuery 方法——执行查询语句 358
5.4.4 executeUpdate 方法——执行更新语句 359
5.4.5 getMetaData 方法——获取 ResultSet 对象列的信息 359
5.4.6 getParameterMetaData 方法——获取参数信息 360

PDF电子书基地 http://dayo1982.400gb.com

5.4.7 setAsciiStream 方法——将指定参数设置为给定输入流 360
5.4.8 setBigDecimal 方法——设置大数字参数值 363
5.4.9 setBinaryStream 方法——设置二进制参数值 363
5.4.10 setBlob 方法——设置 Blob 型参数 366
5.4.11 setBoolean 方法——设置布尔型参数 368
5.4.12 setByte 方法——设置字节型参数 369
5.4.13 setBytes 方法——设置字节数组型参数 369
5.4.14 setCharacterStream 方法——将指定参数设置为给定 Reader 对象 370
5.4.15 setClob 方法——设置 Clob 型参数 371
5.4.16 setDate 方法——设置日期型参数 374
5.4.17 setDouble 方法——设置双精度型参数 374
5.4.18 setFloat 方法——设置单精度型参数 375
5.4.19 setInt 方法——设置整型参数 375
5.4.20 setLong 方法——设置长整型参数 376
5.4.21 setNCharacterStream 方法——将指定参数设置为 Reader 对象 376
5.4.22 setNClob 方法——设置 NClob 类型的参数 378
5.4.23 setNString 方法——设置 String 类型的参数 380
5.4.24 setShort 方法——设置 Short 类型的参数 381
5.4.25 setString 方法——设置 String 类型的参数 381
5.4.26 setTime 方法——设置时间类型的参数 382
5.4.27 setTimestamp 方法——设置参数值为时间戳 383
5.4.28 setURL 方法——设置参数为 URL 384
5.5 ResultSet 接口——结果集接口 385
5.5.1 常量——ResultSet 接口的常量 385
5.5.2 absolute 方法——将光标移动到指定行 385
5.5.3 afterLast 方法——将光标移动到最后一行 386
5.5.4 beforeFirst 方法——将光标移动到第一行之前 386
5.5.5 cancelRowUpdates 方法——取消对当前行所作的更新 387
5.5.6 clearWarnings 方法——清除所有警告 387
5.5.7 close 方法——关闭 ResultSet 对象 388
5.5.8 deleteRow 方法——删除当前行 388

5.5.9 findColumn 方法——获取 Resultset 的列索引 389
5.5.10 first 方法——将光标移动到第一行 390
5.5.11 getAsciiStream 方法——获取指定列的 ASCII 字符流的值 390
5.5.12 getBigDecimal 方法——获取 BigDecimal 型的数据 392
5.5.13 getBinaryStream 方法——获取字节流型的数据 393
5.5.14 getBlob 方法——获取 Blob 型的数据 395
5.5.15 getBoolean 方法——获取布尔型的数据 396
5.5.16 getByte 方法——获取字节型的数据 396
5.5.17 getBytes 方法——获取字节数组的数据 397
5.5.18 getCharacterStream 方法——获取 Reader 型的数据 398
5.5.19 getClob 方法——获取 Clob 型的数据 399
5.5.20 getConcurrency 方法——获取并发模式 400
5.5.21 getCursorName 方法——获取 SQL 光标的名称 401
5.5.22 getDate 方法——获取日期型的数据 402
5.5.23 getDouble 方法——获取双精度型的数据 404
5.5.24 getFetchDirection 方法——获取方向 405
5.5.25 getFetchSize 方法——获取 ResultSet 对象的获取大小 405
5.5.26 getFloat 方法——获取单精度型的数据 406
5.5.27 getHoldability 方法——获取可保存性 407
5.5.28 getInt 方法——获取整型数据 407
5.5.29 getLong 方法——获取长整型数据 408
5.5.30 getMetaData 方法——获取对象的列信息 409
5.5.31 getNCharacterStream 方法——获取 Reader 形式的数据 409
5.5.32 getNClob 方法——获取 NClob 型的数据 411
5.5.33 getNString 方法——获取 String 型的数据 413
5.5.34 getObject 方法——获取 object 型的数据 414
5.5.35 getRow 方法——获取当前编号 415
5.5.36 getShort 方法——获取短整型的数据 415
5.5.37 getStatement 方法——获取 Statement 对象 416
5.5.38 getString 方法——获取 String 类型的数据 417
5.5.39 getTime 方法——获取时间型的数据 418

PDF电子书基地 http://dayo1982.400gb.com

5.5.40 getTimestamp 方法——获取时间戳 420
5.5.41 getType 方法——获取 ResultSet 对象的类型 422
5.5.42 getURL 方法——获取 URL 对象形式的数据 423
5.5.43 getWarnings 方法——获取报告的第一个警告 424
5.5.44 insertRow 方法——插入当前行到数据库中 424
5.5.45 isAfterLast 方法——判断光标是否位于最后一行之后 425
5.5.46 isBeforeFirst 方法——判断光标是否位于第一行之前 425
5.5.47 isClosed 方法——判断 ResultSet 对象是否已经关闭 426
5.5.48 isFirst 方法——判断光标是否位于第一行 426
5.5.49 isLast 方法——判断光标是否位于最后一行 427
5.5.50 last 方法——将光标移动到最后一行 427
5.5.51 moveToCurrentRow 方法——将光标移动到记住的光标位置 427
5.5.52 moveToInsertRow 方法——将光标移动到插入行 428
5.5.53 next 方法——将光标从当前的位置向下移动一行 429
5.5.54 previous 方法——光标移动到上一行 430
5.5.55 relative 方法——按相对行数移动光标 430
5.5.56 setFetchDirection 方法——设置行的处理方向 431
5.5.57 setFetchSize 方法——设置结果集的最大行数 431
5.5.58 updateAsciiStream 方法——用 ascii 流值更新指定的列 432
5.5.59 updateBigDecimal 方法——用 BigDecimal 值更新指定列 437
5.5.60 updateBinaryStream 方法——用二进制流值更新指定的列 438
5.5.61 updateBlob 方法——用 Blob 值更新指定列 443
5.5.62 updateBoolean 方法——用 boolean 值更新指定列 447
5.5.63 updateByte 方法——用 byte 值更新指定列 449
5.5.64 updateBytes 方法——用 byte 数组值更新指定列 450
5.5.65 updateCharacterStream 方法——用二进制流值更新指定的列 452
5.5.66 updateDate 方法——用 Date 值更新指定列 456
5.5.67 updateDouble 方法——用 double 值更新指定列 458
5.5.68 updateFloat 方法——用 float 值更新指定列 459

5.5.69 updateInt 方法——用 int 值更新指定列 460
5.5.70 updateLong 方法——用 long 值更新指定列 462
5.5.71 updateNString 方法——用 String 值更新指定列 463
5.5.72 updateNull 方法——用 null 值更新指定列 464
5.5.73 updateRow 方法——更新当前行 466
5.5.74 updateShort 方法——用 short 值更新指定列 466
5.5.75 updateString 方法——用 String 值更新指定列 468
5.5.76 updateTime 方法——用 Time 值更新指定列 469
5.5.77 updateTimestamp 方法——用时间戳更新指定列 470

第 6 章　JSP 指令 472
6.1 include 指令——包含文件指令 472
6.2 Page 指令——页面指令 472
6.2.1 autoFlush 属性——是否自动输出缓冲区的内容 473
6.2.2 buffer 属性——设置的缓冲区大小 473
6.2.3 language 属性——设置 JSP 脚本使用的语言 473
6.2.4 contentType 属性——设置页面响应的 MIME 类型 474
6.2.5 errorPage 属性——指定异常处理页面 474
6.2.6 extends 属性——指定页面转换为 Servlet 后继承的类 474
6.2.7 import 属性——导入需要用到的包 474
6.2.8 info 属性——设置页面相关信息 475
6.2.9 isErrorPage 属性——设置是否使用 exception 异常对象 475
6.2.10 isIEIgnored 属性——设置是否忽略 EL 表达式 476
6.2.11 isThreadSafe 属性——设置是否以多线程方式处理用户请求 476
6.2.12 pageEncoding 属性——设置字符的编码 476
6.2.13 session 属性——设置当前页面是否支持 session 476
6.3 taglib 指令——标签库指令 476

第 7 章　JSP 的动作标签 478
7.1 <jsp:fallback>——不支持插件时输出提示信息标签 478
7.2 <jsp:forward>——请求转发标签 478
7.3 <jsp:getProperty>——获取属性值标签 479

PDF电子书基地 http://dayo1982.400gb.com

7.4 <jsp:include>——包含文件标签 480
7.5 <jsp:param>——传递单个参数标签 480
7.6 <jsp:params>——传递多个参数标签 481
7.7 <jsp:plugin>——插入 Applet 或 JavaBean 标签 481
7.8 <jsp:setProperty>——设置属性标签 484
7.9 <jsp:useBean>——创建 Bean 实例标签 486

第 8 章 JSP 内置对象 488
8.1 application 对象——应用对象 488
8.1.1 getAttribute(String name)方法——获取属性值 488
8.1.2 getAttributeName 方法——获取属性的初始值 488
8.1.3 getContextPath 方法——获取 Web 应用的路径 489
8.1.4 getMajorVersion 方法——获取 Servlet API 的最大版本号 489
8.1.5 getMimiType 方法——获取指定文件的 MIME 类型 489
8.1.6 getMinorVersion 方法——获取服务器支持的 Servlet API 最小版本号 489
8.1.7 getNamedDispatcher 方法——根据指定参数名返回一个 Request Dispatcher 对象 490
8.1.8 getRealPath(String path)方法——获取虚拟路径的真实路径 490
8.1.9 getRequestDispatcher 方法——根据指定路径名返回一个 Request Dispatcher 对象 490
8.1.10 getResource(Stirng path)方法——获取指定资源的 URL 路径 491
8.1.11 getResourceAsStream 方法——获取指定资源的输入流对象 491
8.1.12 getResourcePaths 方法——获取路径列表 492
8.1.13 log 方法——写入日志信息到日志文件 492
8.1.14 getServletInfo 方法——获取 servlet 容器信息 493
8.1.15 removeAttribute(String key)方法——移除属性 493
8.1.16 setAttribute(String name,Object object)方法——设置属性 493

8.2 config 对象——配置对象 494
8.2.1 getInitParameter 方法——获取单个初始参数的值 494
8.2.2 getInitParameterNames 方法——获取全部初始参数的名字 494
8.2.3 getServletContext 方法——获取 Servlet 上下文 494
8.2.4 getServletName 方法——获取 Servlet 的名字 495
8.3 exception 对象——异常对象 495
8.3.1 fillInStackTrace 方法——在异常堆栈跟踪中填充 495
8.3.2 getCause 方法——返回产生异常的原因 496
8.3.3 getMessage 方法——获取异常消息文本 496
8.3.4 getLocalizedMessage 方法——获取本地化语言的异常信息 497
8.3.5 getStackTrace 方法——获取堆栈跟踪元素的数组 497
8.3.6 initCause 方法——设置初始化的异常信息 498
8.3.7 printStackTrace 方法——输出异常的栈跟踪轨迹 498
8.3.8 setStackTrace 方法——设置堆栈跟踪元素 500
8.3.9 toString 方法——返回异常的简短描述 500
8.4 out 输出对象——输出对象 501
8.4.1 clear 方法——清除缓冲区里的数据 501
8.4.2 clearBuffer 方法——清除缓冲区里的数据 501
8.4.3 close 方法——关闭 JSP 的输出流 501
8.4.4 flush 方法——清空并输出当前缓冲区 502
8.4.5 getBufferSize 方法——获得输出流缓冲区的大小 502
8.4.6 getRemaining 方法——获得缓冲区的可用空间 502
8.4.7 isAutoFlush 方法——判断缓冲区是否自动刷新 503
8.4.8 newLine 方法——写入一个行分隔符 503
8.4.9 print 方法——输出数据到 JSP 页面 503
8.4.10 println 方法——向 JSP 页面输出空字符串和换行符号 507
8.4.11 其他 println 方法 507
8.5 page 对象——页面对象 509
8.5.1 equals 方法——比较两个对象是否相等 509
8.5.2 getClass 方法——获取 page 对象 509
8.5.3 hashCode 方法——返回对象的哈希码值 509

PDF电子书基地 http://dayo1982.400gb.com

8.6 pageContext 对象——页面上下文对象 510
8.6.1 forward 方法——页面转发 510
8.6.2 getErrorData 方法——获取错误信息 510
8.6.3 getAttribute 方法——在指定范围内检索属性 511
8.6.4 getException 方法——获取异常信息 511
8.6.5 getPage 方法——获取当前 page 对象 512
8.6.6 getRequest 方法——获取当前的 request 对象 512
8.6.7 getResponse 方法——获取当前的 response 对象 512
8.6.8 getServletConfig 方法——获取当前页面的 ServletConfig 对象 512
8.6.9 getServletContext 方法——获取 ServletContext 的实例 513
8.6.10 getSession 方法——获取当前的 session 对象 513
8.6.11 handlePageException 方法——抛出异常 513
8.6.12 include 方法——包含其他文件 515
8.6.13 initialize 方法——初始化 PageContext 对象 515
8.6.14 release 方法——释放相关资源 516
8.6.15 removeAttribute 方法——移除指定范围的对象中的属性 516
8.6.16 setAttribute 方法——在指定对象范围中设置属性 517
8.7 request 对象——请求对象 517
8.7.1 getAttribute 方法——获取指定的属性值 518
8.7.2 getAttributeNames 方法——获取 request 对象所有属性的名字集合 518
8.7.3 getAuthType 方法——获取保护 servlet 的认证方案名 518
8.7.4 getCookies 方法——获取所有的 cookie 对象 519
8.7.5 getCharacterEncoding 方法——获取请求页面的字符编码格式 519
8.7.6 getContentLength 方法——获取请求的 Body 的长度 519
8.7.7 getContextPath 方法——获取当前 Web 所在的目录 520
8.7.8 getDateHeader 方法——获取构建 Date 对象的 long 值 520
8.7.9 getHeader 方法——获得 Http 协议定义的文件头信息 520
8.7.10 getHeaders 方法——获取指定请求头的所有值 521
8.7.11 getHeadersNames 方法——获取所有请求头的名字 521

8.7.12 getInputStream 方法——获取请求对象的输入流 522
8.7.13 getIntHeader 方法——获取请求头信息 522
8.7.14 getMethod 方法——获取客户端向服务器端传送数据的方法 522
8.7.15 getParameter 方法——获取客户端传送的参数值 523
8.7.16 getParameterValues 方法——获取指定参数的所有值 523
8.7.17 getPathInfo 方法——获取客户请求 URL 的路径信息 524
8.7.18 getPathTranslated 方法——获取 Servlet 的附加路径信息 524
8.7.19 getProtocol 方法——获取数据传送协议 525
8.7.20 getQueryString 方法——获得查询字符串 525
8.7.21 getRemoteUser 方法——获取请求用户的登录信息 525
8.7.22 getRequestURI 方法——获得客户端相对路径 526
8.7.23 getRequestURL 方法——获得客户端相对路径 526
8.7.24 getRemoteAddr 方法——获取客户端的 IP 地址 526
8.7.25 getRemoteHost 方法——获取客户端的机器名称 527
8.7.26 getRequestedSessionId 方法——获得客户端访问网站的 session ID 值 527
8.7.27 getSession 方法——获取和请求相关的 session 对象 527
8.7.28 getServerName 方法——获取服务器的名字 528
8.7.29 getServerPath 方法——获取请求页面的文件路径 528
8.7.30 getServerPort 方法——获取服务器的端口号 528
8.7.31 getUserPrincipal 方法——返回 Principal 对象 529
8.7.32 isRequestedSessionIdFromCookie 方法——判断 session ID 是否从 Cookie 中获得 529
8.7.33 isRequestedSessionIdFromURL 方法——判断 session ID 是否从 URL 中获得 529
8.7.34 isRequestedSessionIdValid 方法——判断返回 session 是否有效 530
8.7.35 isUserInRole 方法——判断认证后的用户是否属于逻辑的 role 中的成员 530
8.7.36 removeAttribute 方法——删除请求中的一个属性 530
8.7.37 setAttribute 方法——设置 request 对象的属性 531

PDF电子书基地 http://dayo1982.400gb.com

8.8 response 对象——响应对象 531
8.8.1 addCookie 方法——保存客户端的 Cookie 信息 531
8.8.2 addDateHeader 方法——加入对应名称的日期头信息 532
8.8.3 addHeader 方法——添加 Http 文件头信息 532
8.8.4 addIntHeader 方法——加入对应名称的 int 属性 532
8.8.5 containsHeader 方法——判断 Http 文件头是否存在 533
8.8.6 getCharacterEncoding 方法——设置字符编码 533
8.8.7 getOutputStream 方法——获取 JSP 应答输出流 533
8.8.8 sendError 方法——发送错误信息 534
8.8.9 sendRedirect 方法——响应重定向 534
8.8.10 setContentType 方法——设置 MIME 类型 535
8.8.11 setDateHeader 方法——头名称及日期设置为响应头信息 535
8.8.12 setHeader 方法——设置头信息 535
8.8.13 setStatus 方法——设置响应的状态码 536
8.9 session 对象——会话对象 536
8.9.1 getAttribute 方法——获取属性值 536
8.9.2 getAttributeNames 方法——获取属性名称 537
8.9.3 getCreationTime 方法——获取 session 对象创建时间 537
8.9.4 getId 方法——获取惟一的 ID 标识 537
8.9.5 getLastAccessedTime 方法——获取最后发送请求的时间 538
8.9.6 getMaxInactiveInterval 方法——获取 session 有效时间 538
8.9.7 getServletContext 方法——获取 ServletContext 对象 538
8.9.8 invalidate 方法——销毁 session 对象 539
8.9.9 isNew 方法——判断当前用户是否为新用户 539
8.9.10 setAttibute 方法——设置属性值 539
8.9.11 setMaxInactiveInterval 方法——设置 session 有效时间 540
8.9.12 removeAttribute 方法——移除指定属性 540

第 9 章 JSTL 核心标签库 541
9.1 表达式标签 541
9.1.1 <c:out>标签——输出标签 541

9.1.2 <c:set>标签——设置标签 542
9.1.3 <c:remove>标签——移除标签 543
9.1.4 <c:catch>标签——捕获异常标签 543
9.2 流程控制标签 544
9.2.1 <c:if>标签——条件判断标签 544
9.2.2 <c:choose>标签——选择标签 546
9.2.3 <c:when>标签——条件标签 546
9.2.4 <c:otherwise>标签——指定默认处理逻辑标签 547
9.3 循环标签 548
9.3.1 <c:forEach>标签——对对象进行迭代输出标签 548
9.3.2 <c:forTokens>标签——对字符串进行迭代输出标签 549
9.4 URL 操作标签 551
9.4.1 <c:import>标签——文件导入标签 551
9.4.2 <c:redirect>标签——重定向标签 552
9.4.3 <c:url>标签——生成 URL 地址标签 553
9.4.4 <c:param>标签——参数传递标签 554

第 10 章 JSTL 的格式与国际化标签库 556
10.1 <fmt:formatNumber>标签——格式化数字 556
10.2 <fmt:parseNumber>标签——转换为数字 558
10.3 <fmt:formatDate>标签——格式化时间 560
10.4 <fmt:parseDate>标签——转换为时间 561
10.5 <fmt:setTimeZone>标签——设置时区 563
10.6 <fmt:timeZone>标签——格式化时区 563
10.7 <fmt:setBundle>标签——指定消息资源 564
10.8 <fmt:bundle>标签——读取绑定消息资源 565
10.9 <fmt:message>标签——读取本地消息 566
10.10 <fmt:param>标签——指定资源参数 567
10.11 <fmt:setlocale>标签——设置语言区域 568
10.12 <fmt:requestEncoding>标签——设置编码 569

第 11 章 JSTL 的 SQL 标签库 571
11.1 <sql:setDataSource>标签——设置数据源 571

PDF电子书基地 http://dayo1982.400gb.com

11.2 <sql:query>标签——通过 SQL 语句查询 572
11.3 <sql:update>标签——更新数据 575
11.4 <sql:param>标签——指定 SQL 参数 576
11.5 <sql:dateParam>标签——填充日期型参数 577
11.6 <sql:transaction>标签——事务处理 578

第 12 章 JSTL 的 XML 标签库 580
12.1 <x:parse>标签——解析指定的 XML 内容 580
12.2 <x:out>标签——输出 XML 信息 581
12.3 <x:set>标签——保存 XML 节点 583
12.4 <x:if>标签——XPath 条件执行 JSP 代码 584
12.5 <x:choose>标签——完成条件判断 585
12.6 <x:when>标签——根据条件执行代码 585
12.7 <x:otherwise>标签——不满足条件执行 586
12.8 <x:forEach>标签——遍历 XML 文件 587
12.9 <x:transform>标签——XML 到 XSLT 转换 588
12.10 <x:param>标签——定义参数 590

第 13 章 Struts 控制器组件 592
13.1 Action 类——请求处理 592
13.1.1 addErrors 方法——添加指定的错误键 592
13.1.2 addMessages 方法——添加 ActionMessages 592
13.1.3 execute 方法——完成用户请求处理 593
13.1.4 isTokenValid 方法——验证令牌机制 594
13.1.5 getResources 方法——获取默认 MessageResources 595
13.1.6 resetToken 方法——删除令牌属性 595
13.1.7 saveToken 方法——保存新令牌 596
13.1.8 saveErrors 方法——保存错误信息 596
13.1.9 saveMessages 方法——保存 ActionMessages 对象 597
13.2 ActionServlet 类——核心控制器组件 597
13.2.1 doGet 方法——处理用户 HTTP GET 请求 597
13.2.2 doPost 方法——处理用户 HTTP POST 请求 598
13.2.3 destroy 方法——移除控制器组件 598
13.2.4 init 方法——初始化 599

13.2.5 initInternal 方法——初始化 Struts 的消息资源 599
13.2.6 initOther 方法——加载 ActionServlet 初始化参数 600
13.2.7 process 方法——处理接受的请求并做出反应 600
13.3 DispatchAction 类——请求分发 601
13.3.1 execute 方法——处理请求 601
13.3.2 getParameter 方法——获取 parameter 参数值 602
13.3.3 getMethodName 方法——返回方法名 603
13.4 ForwardAction 类——转发请求 603
13.5 IncludeAction 类——包含其他 Web 组件 604
13.6 LookupDispatchAction 类——多业务处理 605
13.6.1 execute 方法——请求处理 605
13.6.2 getKeyMethodMap 方法——返回 Map 集合对象 606
13.6.3 getMethodName 方法——获取方法名 606
13.7 ModuleUtils 类——处理子应用模块 607
13.7.1 getInstance()方法——返回 ModuleUtils 实例 607
13.7.2 getModuleConfig()方法——返回 ModuleConfig 对象 607
13.8 SwitchAction 类——处理多应用模块 608

第 14 章 Struts Bean 标签库 609
14.1 <bean:define>标签——定义一个变量 609
14.2 <bean:cookie>标签——访问 Cookie 对象 611
14.3 <bean:header>标签——访问 HTTP 头信息 611
14.4 <bean:parameter>标签——参数设为变量 612
14.5 <bean:page>标签——隐含对象设为变量 612
14.6 <bean:size>标签——获取集合或数组长度 613
14.7 <bean:message>标签——输出资源信息 614
14.8 <bean:write>标签——输出信息 615
14.9 <bean:resource>标签——检索资源内容 616
14.10 <bean:include>标签——包含 Web 资源 617
14.11 <bean:struts>标签——获取 Struts 对象 618

第 15 章 Struts HTML 标签库 620
15.1 <html:base>标签——获得页面 UTL 620
15.2 <html:checkbox>标签——生成复选框 620

PDF电子书基地 http://dayo1982.400gb.com

15.3 <html:errors>标签——输出错误信息 621
15.4 <html:file>标签——上传文件 622
15.5 <html:form>标签——创建表单 623
15.6 <html:hidden>标签——生成隐藏属性 623
15.7 <html:html>标签——<html>元素 624
15.8 <html:img>标签——嵌入图片 624
15.9 <html:image>标签——生成< img>标签 626
15.10 <html:link>标签——生成<a>标签 627
15.11 <html:messages>标签——输出信息 628
15.12 <html:multibox>标签——生成复选框 629
15.13 <html:option>标签——生成列表选项 630
15.14 <html:options>标签——生成多个列表 631
15.15 <html:optionCollection>标签——下拉列表 632
15.16 <html:password>标签——生成密码框 634
15.17 <html:radio>标签——生成单选按钮 634
15.18 <html:submit>标签——生成提交按钮 635
15.19 <html:select>标签——生成下拉列表框 636
15.20 <html:text>标签——生成文本框 637
15.21 <html:textarea>标签——生成文本域 637

第 16 章 Struts logic 标签库 639
16.1 <logic:equal>标签——比较对象是否相等 639
16.2 <logic:notEqual>标签——比较对象是否不相等 639
16.3 <logic:greaterEqual>标签——比较是否大于或等于 640
16.4 <logic:greaterThan>标签——比较是否大于 641
16.5 <logic:lessEqual>标签——比较小于或等于 642
16.6 <logic:lessThan>标签——比较是否小于 642
16.7 <logic:match>标签——匹配常量 643
16.8 <logic:notMatch>标签——不匹配常量 644
16.9 <logic:empty>标签——对象为空 645
16.10 <logic:notEmpty>标签——对象不为空 646
16.11 <logic:present>标签——是否存在 647
16.12 <logic:notPresent>标签——不存在 648

16.13 <logic:messagesPresent>标签——存在指定消息 649
16.14 <logic:messagesNotPresent>标签——不存在指定消息 650
16.15 <logic:iterate>标签——遍历集合或数组 650
16.16 <logic:forward>标签——请求转发 652
16.17 <logic:redirect>标签——请求重定向 653

第 17 章 Hibernate 常用类与接口 654
17.1 Configuration 类——Hibernate 配置 654
17.1.1 构造方法——Configuration 类的构造方法 654
17.1.2 configure 方法——加载 Hibernate 配置文件 654
17.1.3 addClass 方法——加载实体类 655
17.1.4 buildSessionFactory 方法——构建 SessionFactory 655
17.1.5 addResource 方法——指定实体类的映射文件 656
17.1.6 setProperty 方法——为 Configuation 对象指定配置属性 656
17.2 SessionFactory 接口——Session 工厂 657
17.2.1 openSession 方法——开启一个 Session 657
17.2.2 isClosed 方法——SessionFactory 是否关闭 659
17.2.3 close 方法——关闭 SessionFactory 对象 659
17.3 Session 接口——持久化操作接口 660
17.3.1 beginTransaction 方法——开启一个事务 660
17.3.2 isOpen 方法——判断 Session 是否开启 660
17.3.3 close 方法——关闭 Session 661
17.3.4 save 方法——保存对象 661
17.3.5 update 方法——更新对象 662
17.3.6 saveOrUpdate 方法——保存或更新对象 662
17.3.7 delete 方法——删除对象 663
17.3.8 get 方法——查询对象 664
17.3.9 load 方法——加载对象 666
17.3.10 evict 方法——从 Session 缓存中清除对象 667
17.3.11 clear 方法——清除 Session 缓存中的所有所有对象 668
17.3.12 contains 方法——判断对象是否在缓存中存在 668
17.3.13 createQuery 方法——创建 Query 对象 668

PDF电子书基地 http://dayo1982.400gb.com

17.3.14 createSQLQuery 方法——创建 SQLQuery 对象 669
17.3.15 createCriteria 方法——创建 Criteria 对象 670
17.3.16 lock 方法——指定的锁定模式进行锁定 670
17.4 Transaction 接口——事物接口 671
17.4.1 begin 方法——开启事务 671
17.4.2 commit 方法——提交事务 671
17.4.3 rollback 方法——回滚事务 672
17.4.4 wasCommitted 方法——判断事务是否被提交 672
17.4.5 wasRolledBack 方法——判断事务是否回滚 673
17.5 Query 接口——HQL 查询接口 674
17.5.1 list 方法——返回查询结果的 List 集合 674
17.5.2 iterate 方法——返回查询结果的迭代器 674
17.5.3 uniqueResult 方法——单值检索 675
17.5.4 setMaxResults 方法——设置最大返回记录数 676
17.5.5 setFirsResult 方法——设置结果集的起始点 676
17.5.6 setEntity 方法——用于绑定实体类参数 677
17.5.7 setParameter 方法——绑定任意类型的参数 677
17.5.8 setDate 方法——绑定映射类型为 Date 的参数 678
17.5.9 setString 方法——绑定映射类型为 String 的参数 678
17.5.10 setInteger 方法——绑定映射类型为 int 的参数 679
17.5.11 setBoolean 方法——绑定映射类型为 boolean 的参数 680
17.5.12 setDouble 方法——绑定映射类型为 double 的参数 680
17.5.13 setFloat 方法——绑定映射类型为 float 的参数 681
17.6 Criteria 接口——QBC 查询接口 682
17.6.1 add 方法——设置查询条件 682
17.6.2 addOrder 方法——设置结果集的排序规则 682
17.6.3 createCriteria 方法——创建一个新的 Criteria 对象 683
17.6.4 createAlias 方法——建立连接查询 683
17.6.5 list 方法——返回一个 List 集合 684
17.6.6 setMaxResults 方法——设置最大返回记录数 684
17.6.7 setFirsResult 方法——设置结果集的起始点 685
17.6.8 uniqueResult 方法——单值检索 685
17.6.9 setProjection 方法——设置查询的投影列 685

17.7 Projections 类——帮助 criteria 接口完成分组与统计 686
17.7.1 projectionList 方法——创建多个投影列 686
17.7.2 id 方法——投影对象标识符 687
17.7.3 property 方法——指明投影对象的某个属性 688
17.7.4 distinct 方法——设置查询结果的唯一性 688
17.7.5 groupProperty 方法——设置参加分组的属性列 688
17.7.6 avg 方法——计算某一列的平均值 689
17.7.7 sum 方法——对某一列进行求和 689
17.7.8 max 方法——计算某一列的最大值 690
17.7.9 min 方法——计算某一列的最小值 690
17.7.10 rowCount 方法——满足条件的记录的数目 691
17.7.11 count 方法——返回某个属性的数目 691
17.8 Restrictions 类——创建查询条件 691
17.8.1 allEq 方法——设置一系列的相等条件 691
17.8.2 eq 方法——设置等于条件 692
17.8.3 lt 方法——设置小于条件 692
17.8.4 le 方法——设置小于等于条件 693
17.8.5 gt 方法——设置大于条件 693
17.8.6 ge 方法——设置大于等于条件 693
17.8.7 between 方法——设置在两者之间的条件 694
17.8.8 like 方法——设置模糊查询的条件 694
17.8.9 in 方法——判断是否在给定的数组内 695
17.8.10 and 方法——逻辑与操作 695
17.8.11 or 方法——逻辑或操作 695
17.8.12 not 方法——逻辑非操作 696
17.8.13 isNotNull 方法——判断是否不为 null 696
17.8.14 isNull 方法——判断是否为 null 697
17.8.15 sqlRestriction 方法——通过 SQL 语句限定查询条件 697
17.9 SQLQuery 接口——SQL 查询接口 697
17.9.1 addEntity 方法——将别名和实体类联系 698
17.9.2 addJoin 方法——实体和集合相关联 700
17.9.3 addScalar 方法——返回标量和实体的结合体 701

PDF电子书基地 http://dayo1982.400gb.com

第 18 章 Hibernate 配置属性 702
18.1 dialect 属性——数据库方言 702
18.2 connection.driver_class 属性——数据库驱动 703
18.3 connection.url 属性——数据库连接 URL 705
18.4 connection.username 属性——数据库连接用户名 706
18.5 connection.password 属性——数据库连接密码 707
18.6 show_sql 属性——显示 SQL 语句 708
18.7 format_sql 属性——格式化 SQL 语句 708
18.8 use_sql_comments 属性——输出注释信息 709
18.9 default_schema 属性——默认的表空间 710
18.10 default_catalog 属性——默认的表目录 710
18.11 session_factory_name 属性——绑定到 JNDI 711
18.12 max_fetch_depth 属性——设置抓取深度 712
18.13 default_batch_fetch_size 属性——设置默认抓取数量 713
18.14 default_entity_mode 属性——默认实体表现模式 713
18.15 order_updates 属性——为 SQL 更新排序 714
18.16 generate_statistics 属性——收集统计数据 715
18.17 use_identifer_rollback 属性——重设默认值 715
18.18 cache.use_query_cache 属性——判断是否缓存查询结果 716
18.19 cache.use_second_level_cache 属性——判断是否关闭二级缓存 717
18.20 cache.provider_class 属性——设置二级缓存实现类的全名 718
18.21 cache.region_prefix 属性——设置二级缓存的前缀名称 718
18.22 cache.use_minimal_puts 属性——判断是否优化二级缓存 719
18.23 transaction.factory_class 属性——设置事务管理类型 720

第 19 章 Hibernate 映射节点 721
19.1 <hibernate-mapping>节点——实体映射 721
19.2 <class>节点——定义类/表的映射关系 722
19.3 <id>节点——标识字段 724
19.4 <generator>节点——主键生成策略 726
19.5 <key>节点——映射外键 727
19.6 <property >节点——映射实体属性 728

19.7 <one-to-one>节点——一对一关联关系 730
19.8 <many-to-many>节点——多对多关联关系 731
19.9 <set>节点——映射集合 732
19.10 <many-to-one>节点——多对一关联关系 733
19.11 <one-to-many>节点——一对多关联关系 735

第 20 章 Spring 容器 736
20.1 ClassPathXmlApplicationContext 类——创建容器 736
20.2 FileSystemXmlApplicationContext 类——指定配置文件的位置 737
20.3 XmlBeanFactory 类——解析 Spring 的配置 738
20.4 WebApplicationContext 接口——初始化 Spring 容器 738
20.5 Spring 容器类的方法——容器类 742
20.5.1 containsBean 方法——判断是否包含指定名称 JavaBean 742
20.5.2 getBean 方法——获取容器中的 JavaBean 743
20.5.3 getBeanDefinitionCount 方法——获取 JavaBean 数量 743
20.5.4 getBeanDefinitionNames 方法——获取 JavaBean 名称 744
20.5.5 getBeanNamesForType 方法——通过类型获取 Bean 名称 744
20.5.6 getBeansOfType 方法——获取指定类型的 JavaBean 745
20.5.7 getType 方法——获取 JavaBean 的类型 745
20.5.8 isSingleton 方法——判断是否定义 Singleton 模式 746
20.5.9 isTypeMatch 方法——判断 JavaBean 是否匹配指定的类型 746
20.5.10 getServletContext 方法——获取 ServletContext 上下文 747

第 21 章 Spring 持久化 748
21.1 DriverManagerDataSource 类——驱动管理器数据源 748
21.1.1 构造方法——DriverManagerDataSource 类的构造方法 748
21.1.2 getConnection 方法——获取数据库连接对象 750
21.1.3 getConnectionProperties 方法——获取数据库连接属性 751
21.1.4 getDriverClassName 方法——获取数据库驱动类的名称 752
21.1.5 getPassword 方法——获取数据库的密码 752

PDF电子书基地 http://dayo1982.400gb.com

21.1.6 getUrl 方法——获取数据库的 URL 路径 753
21.1.7 getUsername 方法——获取数据库的用户名称 753
21.1.8 setConnectionProperties 方法——设置数据库连接属性 753
21.1.9 setDriverClassName 方法——设置数据源的驱动类 754
21.1.10 setPassword 方法——设置数据库的密码 755
21.1.11 setUrl 方法——设置连接数据库的 URL 路径 755
21.1.12 setUsername 方法——设置访问数据库的用户名 755
21.2 HibernateTemplate 类——Hibernate 模板 755
21.2.1 构造方法——HibernateTemplate 类的构造方法 756
21.2.2 delete 方法——删除的实体对象 756
21.2.3 deleteAll 方法——删除指定集合中实体对象 757
21.2.4 execute 方法——接口回调执行操作 757
21.2.5 executeFind 方法——接口回调查询数据 758
21.2.6 find 方法——执行 Hibernate 查询 759
21.2.7 get 方法——获取指定 ID 编号的实体对象 760
21.2.8 loadAll 方法——装载指定类型的所有实体对象 760
21.2.9 save 方法——保存实体对象 761
21.2.10 saveOrUpdate 方法——保存或更新实体对象 761
21.2.11 saveOrUpdateAll 方法——保存或更新实体对象集合 762
21.2.12 update 方法——更新实体对象 763
21.3 JdbcTemplate 类——JDBC 模板 764
21.3.1 batchUpdate 方法——批量更新语句 764
21.3.2 query 方法——用于 JDBC 查询 764
21.3.3 queryForInt 方法——int 类型单值查询 767
21.3.4 queryForList 方法——返回结果集合的 SQL 查询 768
21.3.5 queryForLong 方法——long 类型单值查询方法 770
21.3.6 queryForMap 方法——查询结果的 Map 集合 771
21.3.7 queryForObject 方法——查询的结果以对象类型返回 772
21.3.8 update 方法——更新数据库中的数据 772

第 22 章 Spring Web 表单标签 774
22.1 <form:checkbox>——复选框元素 774
22.2 <form:errors>——验证错误信息 776

22.3 <form:form>——定义 HTML 表单 777
22.4 <form:hidden>——定义表单隐藏组件 777
22.5 <form:input>——定义文本框 778
22.6 <form:option>——定义下拉选择框 778
22.7 <form:options>——下拉框或列表框 779
22.8 <form:password>——定义密码框 781
22.9 <form:radiobutton>——定义单选按钮 782
22.10 <form:select>——下拉框或列表框 783
22.11 <form:textarea>——定义文本域 784

附录　字母索引 786

PDF电子书基地 http://dayo1982.400gb.com

第1章 字 符 串

java.lang 包中的 String 类代表字符串，例如，字符串“abc”。Java 中将字符串作为对象来处理。String 类提供了各种处理字符串的方法。

1.1 charAt 方法——提取指定字符

该方法返回指定索引位置的 char 值。索引范围为 0～length()−1。序列的第一个 char 值位于索引 0 处，第二个位于索引 1 处，依此类推，这类似于数组索引。如果索引指定的 char 值是代理项，则返回代理项值。

方法声明

```
public char charAt(int index)
```

入口参数：index 是 char 的索引值。

返回值：此字符串指定索引处的 char 值。第一个 char 值位于索引 0 处。

示例

本示例使用 charAt 函数获取字符串 strCom 中索引为 4 的 char 值，并将结果赋值给 int 变量 strLower。

```
String strCom = "I LIKE JAVA";
int strLower = strCom.charAt(4);
```

示例

本示例使用 charAt 函数获取字符串 strCom 中索引为 2 的 char 值，并将结果赋值给 int 变量 str。

```
String strCom = "明日科技";
int str = strCom.charAt(2);
```

1.2 codePointAt 方法——提取索引字符代码点

返回指定索引处的字符（Unicode 代码点）。索引引用 char 值（Unicode 代码单元），其范围为 0～length()−1。

如果给定索引指定的 char 值属于高代理项范围，则后续索引小于此 String 的长度；如果后续索引处的 char 值属于低代理项范围，则返回该代理项对相应的增补代码点。否则，返回给定索引处的 char 值。

方法声明

```
public int codePointAt(int index)
```

入口参数：index 是 char 的索引值。

返回值：在 index 处字符的代码点值。

抛出异常

IndexOutOfBoundsException：如果 index 参数为负或小于此字符串的长度。

示例

本示例使用 codePointAt 方法获取字符串 strCom 中索引为 8 的字符(Unicode 代码点)，并将结果赋值给 int 变量 strLower。

```
String strCom = "I LIKE JAVA";
int strLower = strCom.codePointAt(8);
```

1.3 codePointBefore 方法——获取索引前一个字符的代码点

返回指定索引之前的字符（Unicode 代码点）。索引引用 char 值（Unicode 代码单元），其范围为 1～length。

如果（index-1）处的 char 值属于高代理项范围，则（index-2）为非负；如果 index-2 处的 char 值属于高代理项范围，则返回该代理项对的增补代码点值。如果 index-1 处的 char 值是未配对的低（高）代理项，则返回代理项值。

方法声明

```
public int codePointBefore(int index)
```

入口参数：index 是应返回的代码点之后的索引。

返回值：给定索引前面的 Unicode 代码点。

抛出异常

IndexOutOfBoundsException：如果 index 参数小于 1 或大于此字符串的长度。

示例

本示例使用 codePointBefore 方法获取字符串 strCom 中索引为 8 之前的字符（Unicode 代码点），并将结果赋值给 int 变量 strLower。

```
String strCom = "I LIKE JAVA";
int strLower = strCom. codePointBefore (8);
```

1.4 codePointCount 方法——获取指定范围文本代码点数

返回此 String 的指定文本范围中的 Unicode 代码点数。文本范围始于指定的 beginIndex，一直到索引 endIndex-1 处的 char。因此，该文本范围的长度（用

PDF电子书基地 http://dayo1982.400gb.com

char 表示）是 endIndex-beginIndex。该文本范围内每个未配对的代理项计为一个代码点。

方法声明

```
codePointCount(int beginIndex, int endIndex)
```

入口参数

beginIndex：文本范围的第一个 char 的索引。

EndIndex：文本范围的最后一个 char 之后的索引。

返回值：指定文本范围中 Unicode 代码点的数量。

抛出异常

IndexOutOfBoundsException：如果 beginIndex 为负，或 endIndex 大于此 String 的长度，或 beginIndex 大于 endIndex。

示例

本示例使用 codePointBefore 方法获取字符串 strCom 中索引为 8 之前的字符（Unicode 代码点），并将结果赋值给 int 变量 strLower。

```
String strCom = "I LIKE JAVA";
int strLower = strCom.codePointCount(2,3);
```

1.5 compareTo 方法——比较两个字符串

按字典顺序比较两个字符串。该比较基于字符串中各个字符的 Unicode 值。按字典顺序将此 String 对象表示的字符序列与参数字符串所表示的字符序列进行比较。如果按字典顺序此 String 对象位于参数字符串之前，则比较结果为一个正整数。如果按字典顺序此 String 对象位于参数字符串之后，则比较结果为一个负整数。如果这两个字符串相等，则比较结果为 0；compareTo 方法只在 equals(Object)返回 true 时才返回 0。

这是字典排序的定义。如果这两个字符串不同，那么它们要么在某个索引处的字符不同（该索引对二者均为有效索引），要么长度不同，或者同时具备这两种情况。如果它们在一个或多个索引位置上的字符不同，假设 k 是这类索引的最小值；则在位置 k 上具有较小值的那个字符串（使用<运算符确定），其字典顺序在其他字符串之前。在这种情况下，compareTo 返回这两个字符串在位置 k 处两个 char 值的差，即值：this.charAt(k)-anotherString.charAt(k)。

如果没有字符不同的索引位置，则较短字符串的字典顺序在较长字符串之前。在这种情况下，compareTo 返回这两个字符串长度的差，即值：this.length() -anotherString.length()。

方法声明

```
public int compareTo(String anotherString)
```

入口参数：anotherString 是指定比较的 String。

返回值：如果参数字符串等于此字符串，则返回值 0；如果此字符串按字典顺序小于字符串参数，则返回一个小于 0 的值；如果此字符串按字典顺序大于字符串参数，则返回一个大于 0 的值。

示例

本示例使用 compareTo 函数按字典顺序比较字符串 strCom 和 strCom1，并将结果赋值给 int 变量 strLower。由于字符串中从第 6 个字符开始，字符 p 按字典排序在字符 J 之后，所以字符串 strLower 的返回值为-6。

```
String strCom = "I LIKE JAVA";
String strCom1 = "I LIKE PHP";
int strLower = strCom.compareTo(strCom1);
```

示例

本函数区分字符的大小写，同样使用 compareTo 函数按字典顺序比较字符串 strCom 和 strCom1，并将结果赋值给 int 变量 strLower。这里将比较字符串 strCom1 变为小写，字符串 strLower 的返回值发生了变化，为-32。

```
String strCom = "I LIKE JAVA";
String strCom1 = "i like PHP";
int strLower = strCom.compareTo(strCom1);
```

1.6 compareToIgnoreCase 方法——忽略大小写比较字符串

按字典顺序比较两个字符串，不考虑大小写。此方法返回一个整数。

方法声明

```
public int compareToIgnoreCase(String str)
```

入口参数：str 是指定比较的 String。

返回值：根据指定 String 大于、等于还是小于此 String（不考虑大小写），分别返回负整数、0 或正整数。

示例

本示例使用 compareToIgnoreCase 函数按字典顺序比较字符串 strCom 和 strCom1，并将结果赋值给 int 变量 strLower。虽然比较的字符串一个是小写，一个是大写，但是字符串 strLower 的返回值为 0。

```
String strCom = "I LIKE JAVA";
String strCom1 = "i like java";
int strLower = strCom.compareToIgnoreCase(strCom1);
```


PDF电子书基地 http://dayo1982.400gb.com

1.7 concat 方法——字符串结尾连接

将指定字符串连接到此字符串的结尾。如果参数字符串的长度为 0，则返回此 String 对象；否则，创建一个新的 String 对象，用来表示由此 String 对象表示的字符序列和参数字符串表示的字符序列连接而成的字符序列。

方法声明

```
public String concat(String str)
```

入口参数：str 是连接到此 String 结尾的 String。

返回值：一个字符串，它表示在此对象字符后连接字符串参数字符而成的字符。

示例

本示例使用 concat 函数连接字符串 strCom 和 strCom2，并将结果赋值给 String 变量 str。

```
String strCom = "I LIKE ";
String strCom2 = "JAVA";
String str = strCom.concat(strCom2);
```

示例

本示例使用 concat 函数连接字符串 strCom 和 strCom2，设置字符串 strCom2 的长度为 0，将结果赋值给 String 变量 str。

```
String strCom = "I LIKE Java！ ";
String strCom2 = "";
String str = strCom.concat(strCom2);
```

1.8 contains 方法——判断是否包含指定字符

判断指定的字符串中是否包含指定的 char 值序列，将判断结果以 boolean 类型返回。

方法声明

```
public boolean contains(CharSequence s)
```

入口参数：s 是指定要搜索的序列。

返回值：如果此字符串包含 s，则返回 true，否则返回 false。

示例

本示例使用 contains 函数判断字符串 strCom 中是否包含“JAVA”，并将返回结果赋给 boolean 变量 str。

```
String strCom = "I LIKE JAVA!";
boolean str = strCom.contains("JAVA");
```

1.9 copyValueOf 方法——字符数组生成字符串

返回指定数组中表示该字符序列的 String。

方法声明 1

```
public static String copyValueOf(char[] data)
```

入口参数：data 为字符数组。

返回值：一个 String，它包含字符数组的字符。

注意

该方法返回一个 String，它包含字符数组的字符。可以理解为将一个指定的数组转换成字符串。

示例

本示例使用 copyValueOf 函数将指定的数组转换成字符串，并将返回结果赋给 String 变量 str。

```
public static void main(String[] args) {
    char[] array={'明','日','科','技'};
    String str=String.copyValueOf(array);
    System.out.println(str);
}
```

方法声明 2

返回一个 String，它包含字符数组的指定子数组字符。

```
public static String copyValueOf(char[] data, int offset, int count)
```

入口参数

data：字符数组。

offset：子数组的初始偏移量。

count：子数组的长度。

返回值：一个 String，它包含字符数组的指定子数组的字符。

示例

本示例使用 copyValueOf 函数，获取指定数组中从第 2 个字符开始的 2 个字符，并将返回结果赋给 String 变量 str。

```
public static void main(String[] args) {
    char[] array={'明','日','科','技'};
```


PDF电子书基地 http://dayo1982.400gb.com

```
        String str=String.copyValueOf(array,2,2);
        System.out.println(str);
    }
```

1.10 endsWith 方法——判断后缀字符串

测试此字符串是否以指定的后缀结束，将测试结果以 boolean 类型返回。

方法声明

```
public boolean endsWith(String suffix)
```

入口参数：suffix 为指定的后缀。

返回值：如果参数表示的字符序列是此对象表示的字符序列的后缀，则返回 true；否则返回 false。注意，如果参数是空字符串，或者等于此 String 对象（用 equals(Object) 方法确定），则结果为 true。

示例

本示例使用 endsWith 函数，判断指定的文件是否以.java 为后缀，如果是则返回 true，否则返回 false。

```
String strCom = "string.java";              //定义一个文件名字符串
boolean str = strCom.endsWith(".java"); //判断字符串是否以.java 为后缀
System.out.println(str);
```

1.11 equals 方法——判断字符串相等

该方法用于将字符串与指定的对象相比较。将比较结果以 boolean 类型返回。

方法声明

```
public boolean equals(object obj)
```

入口参数：obj 为进行比较的对象。

返回值：如果给定对象表示的 String 与此 String 相等，则返回 true；否则返回 false。

示例

本示例使用 equals 函数来判断字符串 strCom1 与字符串 strCom2 是否相等。如果两个字符串相等则 boolean 类型变量 strB 返回 true，否则返回 false。

```
String strCom1 = "MN";
String strCom2 = "mn";
boolean strB = strCom1.equals(strCom2);
```

1.12 equalsIgnoreCase 方法——忽略大小写判断字符串相等

该方法将一个字符串与另一个字符串进行比较，在比较过程中忽略大小写。

方法声明

```
public boolean equalsIgnoreCase(String anotherString)
```

入口参数：anotherString 为进行比较的字符串对象。

返回值：如果参数不为 null，且这两个 String 相等（忽略大小写），则返回 true；否则返回 false。

注意

该方法不区分字符串大小写。

示例

本示例使用 equalsIgnoreCase 函数比较字符串 strCom1 与字符串 strCom2 是否相等。

```
String strCom1 = "MN";
String strCom2 = "mn";
boolean strB = strCom1.equalsIgnoreCase(strCom2);
```

1.13 format 方法——格式化字符串

使用指定的格式字符串和参数返回一个格式化字符串。

方法声明 1

```
public static String format(String format,Object... args)
```

入口参数

format：格式字符串。

args：格式字符串中由格式说明符引用的参数。参数数目是可变的，可以为 0。

常规类型的格式化可应用于任何参数类型，可通过如下表所示的转换符来实现。

常规转换符的说明及示例

转 换 符	说 明	示 例
%b、%B	格式化为布尔类型	false
%h、%H	格式化为散列码	A05A5198
%s、%S	格式化为字符串类型	“abc”
%c、%C	格式化为字符类型	‘w’
%d	格式化为十进制整数	26
%o	格式化为八进制整数	12

PDF电子书基地 http://dayo1982.400gb.com

续表

转 换 符	说 明	示 例
%x、%X	格式化为十六进制整数	4b1
%e	格式化为用计算机科学记数法表示的十进制数	1.700000e+01
%a	格式化为带有效位数和指数的十六进制浮点值	0X1.C000000000001P4
%n	结果为特定于平台的行分隔符	
%%	结果为字面值'%'	%

返回值：一个格式化字符串。

示例

本示例将“400/2”返回结果类型转换为字符串，并将“3>5”返回结果格式化为布尔类型。

```
String str = String.format("%d",400/2);
String str2 = String.format("%b",3 > 5);
```

示例

本示例将“15.000”转换为科学计数法表示的十进制整数，并将“15”返回，结果类型转换为十六进制整数。

```
String str = String.format("%e",15.000);          //格式化为科学计数法表示的十进制整数
String str2 = String.format("%x",15);             //格式化为十六进制
```

方法声明 2

通过给定的特殊转换符作为参数来实现对日期和时间字符串的格式化。

```
public static String format(Locale l, String format,Object... args)
```

入口参数

l：格式化过程中要应用的语言环境。如果 l 为 null，则不进行本地化。

format：格式字符串。

args：格式字符串中由格式说明符引用的参数。如果还有格式说明符以外的参数，则忽略这些额外的参数。参数的数目是可变的，可以为 0。参数的最大数目受 Java Virtual Machine Specification 所定义的 Java 数组最大维度的限制。有关 null 参数的行为依赖于转换。

format()方法常用的日期和时间的格式转换符如下表所示。

常见的日期时间转换符的说明和示例

转 换 符	说 明	示 例
%te	一个月中的某一天（1～31）	12
%tb	指定语言环境的月份简称	Jan（英文）、一月（中文）
%tB	指定语言环境的月份全称	February（英文）、二月（中文）
%tA	指定语言环境的星期几全称	Monday（英文）、星期一（中文）

续表

转 换 符	说 明	示 例
%ta	指定语言环境的星期几简称	Mon（英文）、星期一（中文）
%tc	包括全部日期和时间信息	星期三 十月 25 13:37:22 CST 2008
%tY	4位年份	2008
%tj	一年中的第几天（001～366）	060
%tm	月份	05
%td	一个月中的第几天（01～31）	07
%ty	二位年份	08

返回值：一个格式化字符串。

示例

本示例将当前日期以4位年份、月份全称、两位日期形式输出。

```
package strdatetime;
import java.util.Date;
import java.util.Locale;
public class Main {
    public static void main(String[] args) {
        Date date = new Date();
        Locale form=Locale.US;
        String year = String.format(form, "%tY", date);
        String month = String.format(form, "%tB", date);
        String day = String.format(form, "%td", date);
        System.out.println("今年是： " + year + "年");
        System.out.println("现在是： " + month);
        System.out.println("今天是： " + day + "号");
    }
}
```

本示例的运行结果如下。

```
今年是：2008 年
现在是：December
今天是：05 号
```

示例

本示例将当前日期以2位年份、月份简称、全部日期和时间形式输出。

```
public static void main(String[] args) {
    Date date=new Date();
    String year = String.format("%ty", date);          //2 位年份
    String month = String.format("%tb", date);         //月份全称
    String day = String.format("%tc", date);           //全部日期和时间格式
    System.out.println("今年是： " + year + "年");
    System.out.println("现在是： " + month);
    System.out.println("今天是： " + day + "号");
}
```


PDF电子书基地 http://dayo1982.400gb.com

本示例的结果如下：

```
今年是：08 年
现在是：十一月
今天是：星期三 十一月 26 17:05:31 CST 2008 号
```

1.14 getBytes 方法——获取字符串的字节数组

使用平台的默认字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

方法声明 1

```
public byte[] getBytes();
```

返回值：所得 byte 数组。

示例

本示例使用 getBytes 函数将字符串 strCom 编码为 byte 序列，将返回值赋给 str 数组，并输出返回值。

```
String strCom = "java";              //定义一个字符串
byte[] str = strCom.getBytes();      //将指定字符串编码为 byte 序列
for(int i=0;i<str.length;i++){       //输出返回数组
    System.out.println(str[i]);
}
```

方法声明 2

```
public byte[] getBytes(Charset charset)
```

入口参数：charset 为用于编码 String 的 Charset。

返回值：所得 byte 数组。

抛出异常

UnsupportedEncodingException：如果指定的字符集不受支持。

示例

本示例使用 getBytes 方法，将字符串 strCom 用 Java 虚拟机的默认编码为 byte 序列，将返回值赋给 str 数组，并输出返回值。

```
String strCom = "java";                  //定义一个字符串
//将指定字符串用 Java 虚拟机的默认编码为 byte 序列
byte[] str = strCom.getBytes(Charset.defaultCharset());
for(int i=0;i<str.length;i++){           //输出返回数组
    System.out.println(str[i]);
}
```

方法声明 3

```
public byte[] getBytes(String charsetName)
```

入口参数：charsetName 为受支持的 charset 名称。

返回值：所得 byte 数组。

示例

本示例使用 getBytes 方法，将字符串 strCom 用 GBK 编码为 byte 序列，将返回值赋给 str 数组，并输出返回值。

```
String strCom = "java"; // 定义一个字符串
byte[] str;
try {
        // 将指定字符串用 GBK 编码为 byte 序列
        str = strCom.getBytes("GBK");
        for (int i = 0; i < str.length; i++) { // 输出返回数组
            System.out.println(str[i]);
        }
} catch (UnsupportedEncodingException e) {
        e.printStackTrace();
}
```

1.15 getChars 方法——获取字符数组

将字符从此字符串复制到目标字符数组。

要复制的第一个字符位于索引 srcBegin 处；要复制的最后一个字符位于索引 srcEnd-1 处（因此要复制的字符总数是 srcEnd-srcBegin）。

方法声明

```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

入口参数

srcBegin：字符串中要复制的第一个字符的索引。

srcEnd：字符串中要复制的最后一个字符之后的索引。

dst：目标数组。

dstBegin：目标数组中的起始偏移量。

抛出异常

如果下列任何一项为 true，则抛出 IndexOutOfBoundsException 异常。

- srcBegin 为负。
- srcBegin 大于 srcEnd。
- srcEnd 大于此字符串的长度。
- dstBegin 为负。
- dstBegin+(srcEnd-srcBegin)大于 dst.length。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例使用 getChars 方法，将字符串 strCom 索引 0～10 从 dst 数组的第 0 个位置开始，依次复制到 dst 中。

```
String strCom = "I LIKE JAVA"; // 定义一个字符串
char[] dst = new char[10]; // 定义一个字符数组
strCom.getChars(0, 10, dst, 0); // 将字符串复制到数组
for (int i = 0; i < dst.length; i++) {
        System.out.print(dst[i]);
}
```

1.16 hashCode 方法——生成字符串哈希表

返回此字符串的哈希码。String 对象的哈希码根据以下公式计算：

$$s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + ... + s[n-1]$$

使用 int 算法，这里 s[*i*]是字符串的第 *i* 个字符，*n* 是字符串的长度，^表示求幂（空字符串的哈希码为 0）。

方法声明

```
public int hashCode()
```

返回值：返回此字符串的哈希码。

示例

本示例使用 getChars 方法，输出字符串 strCom 的哈希码。

```
String strCom = "I LIKE JAVA"; // 定义一个字符串
System.out.println(strCom.hashCode());
```

示例

本示例使用 getChars 方法，输出字符串 strCom 的哈希码。

```
String strCom = "明日科技"; // 定义一个字符串
System.out.println(strCom.hashCode());
```

1.17 indexOf 方法——获取字符第一个索引

返回指定字符在此字符串中第一次出现处的索引。如果在此 String 对象表示的字符序列中出现值为 ch 的字符，则返回第一次出现该字符的索引（以 Unicode 代码单元表示）。如果未出现该字符，则返回–1。

方法声明 1

```
public int indexOf(int ch)
```

入口参数：ch 是一个字符（Unicode 代码点）。

返回值：在此对象表示的字符序列中第一次出现该字符的索引；如果未出现

该字符，则返回–1。

示例

本示例使用 indexOf 函数查找数字 5 在字符串 strCom1 中的索引位置，并将返回结果赋值给 int 型变量 index。由于字符串 strCom 中并不包含数字 5，因此 index 的值为–1。

```
String strCom = "I like java";
int index = strCom.indexOf(5);
```

方法声明 2

返回值为返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。在此对象表示的字符序列中第一次出现的大于或等于 fromIndex 的字符的索引；如果未出现该字符，则返回–1。

```
public int indexOf(int ch,int fromIndex)
```

入口参数

ch：一个字符（Unicode 代码点）。

fromIndex：开始搜索的索引位置。

返回值：在此对象表示的字符序列中第一次出现的大于或等于 fromIndex 的字符的索引；如果未出现该字符，则返回–1。

注意

fromIndex 的值没有限制。如果它为负，则与它为 0 的效果同样：将搜索整个字符串。如果它大于此字符串的长度，则与它等于此字符串长度的效果相同：返回–1。

示例

本示例使用 indexOf 函数在字符串 strCom 中搜索字符 7，开始搜索的索引位置是 0，并将返回结果赋值给 int 型变量 index。由于在指定的搜索范围内不包含字符 7，因此 index 的值为–1。

```
String strCom = "I like java";
int index = strCom.indexOf(7,0);
```

方法声明 3

返回指定子字符串在此字符串中第一次出现处的索引。如果它不作为一个子字符串出现，则返回–1。

```
public int indexOf(String str)
```

入口参数：str 是任意字符串。

返回值：如果字符串参数作为一个子字符串在此对象中出现，则返回第一个这种子字符串的第一个字符的索引；如果它不作为一个子字符串出现，则返回–1。

PDF电子书基地 http://dayo1982.400gb.com

注意

返回的整数是 this.startsWith(str, k)为 true 的最小 k 值。

示例

本示例使用 indexOf 函数获取指定字符串“明日科技”在字符串 strCom 中第一次出现的索引位置，并将返回结果赋值给 int 型变量 index。

```
String strCom = "吉林省明日科技有限公司！";
int index = strCom.indexOf("明日科技");
```

方法声明 4

返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始；如果未出现该字符，则返回-1。

```
public int indexOf(String str, int fromIndex)
```

入口参数

str：要搜索的子字符串。

fromIndex：开始搜索的索引位置。

返回值：指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。.

注意

返回的整数是满足下式的最小 k 值：k>= Math.min(fromIndex, this.length()) && this.startsWith(str, k)，如果不存在这样的 k 值，则返回-1。

示例

本示例使用 indexOf 函数获取指定字符串“明日科技”在字符串 strCom 中第一次出现的索引位置，指定开始搜索的索引位置为 5，并将返回结果赋值给 int 型变量 index。

```
String strCom = "吉林省明日科技有限公司！";
int index = strCom.indexOf("明日科技",5);
```

1.18 intern 方法——获取规范化字符串

返回字符串对象的规范化表示形式。

一个初始为空的字符串池，它由类 String 私有地维护。

方法声明

```
public String intern()
```

返回值：一个字符串，内容与此字符串相同，但一定取自具有惟一字符串的池。

注意

当调用 intern 方法时，如果池已经包含一个等于此 String 对象的字符串（用

equals(Object)方法确定），则返回池中的字符串。否则，将此 String 对象添加到池中，并返回此 String 对象的引用。

它遵循以下规则：对于任意两个字符串 s 和 t，当且仅当 s.equals(t)为 true 时，s.intern() == t.intern()才为 true。

所有字面值字符串和字符串赋值常量表达式都使用 intern 方法进行操作。

示例

本示例使用 intern 方法，将 strCom 和 strCom2 进行比较。因为 strCom.equals (strCom2)返回 true，所以 strCom.intern() == strCom2.intern()也返回 true。

```
String strCom = new String("I LIKE JAVA"); // 定义一个字符串
String strCom2 = new String("I LIKE JAVA"); // 定义一个字符串
System.out.println(strCom.equals(strCom2));
System.out.println(strCom == strCom2);
System.out.println(strCom.intern() == strCom2.intern());
```

1.19 isEmpty 方法——判断字符串是否为空

判断字符串的长度是否为空，如果字符串长度为 0，则返回 true，否则返回 false。

方法声明

```
public boolean isEmpty();
```

返回值：如果 length()为 0，则返回 true；否则返回 false。

示例

本示例使用 isEmpty()函数获取字符串 strCom 的值，将返回值赋给 boolean 变量 str。

```
String strCom = "I like java! ";        //定义一个字符串
boolean str = strCom.isEmpty();         //判断字符串是否为空
```

1.20 lastIndexOf 方法——获取字符最后的索引

返回指定字符最后一次出现在指定字符串中的索引位置。如果未找到该字符，则返回-1。

方法声明 1

```
public int lastIndexOf(int ch);
```

入口参数：ch 为一个字符（Unicode 代码点）。

返回值：此对象表示的字符序列中最后一次出现该字符的索引；如果未出现该字符，则返回-1。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例使用 lastIndexOf 函数在字符串 strLast 中搜索字符“a”最后一次出现的索引位置，并将返回结果赋值给 int 型变量 index。

```
String strCom = "abcdefg gfdecba";
int index = strCom.lastIndexOf("a");
```

方法声明 2

返回指定字符在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。如果找到返回值，返回出现处的下标；否则返回-1。

```
public int lastIndexOf(int ch,int fromIndex)
```

入口参数

ch：一个字符（Unicode 代码点）。

fromIndex：开始搜索的索引位置。fromIndex 的值没有限制。如果它大于等于此字符串的长度，则与它小于此字符串长度减 1 的效果相同，将搜索整个字符串。如果它为负，则与它为-1 的效果相同：返回-1。

返回值：在此对象表示的字符序列（小于等于 fromIndex）中最后一次出现该字符的索引；如果在该点之前未出现该字符，则返回-1。

示例

本示例使用 lastIndexOf 函数在字符串 strCom 中搜索字符“a”，从索引位置 5 处开始反向搜索，并将返回结果赋值给 int 型变量 index。由于反向搜索字符“a”的索引位置是 0，因此 index 的值为 0。

```
String strCom = "abcdefg gfdecba";
int index = strCom.lastIndexOf("a",5);
```

方法声明 3

返回指定子字符串在此字符串中最后一次出现处的索引。将最后的空字符串""视为出现在索引值 this.length()处。返回的索引是 this.startsWith(str, k)为 true 的最大 k 值。如果不存在该子字符串，则返回-1。

```
public int lastIndexOf(String str);
```

入口参数：str 是要搜索的子字符串。

返回值：如果字符串参数作为一个子字符串在此对象中出现一次或多次，则返回最后一个该子字符串的第一个字符。如果它不作为一个子字符串出现，则返回-1。

示例

本示例使用 lastIndexOf 函数搜索字符串“明日”在字符串 strCom 中最后出现的位置，并将返回结果赋值给 int 型变量 index。

```
String strCom = "明日科技，明日编程词典，明日软件";
int index = strCom.lastIndexOf("明日");
```

方法声明 4

返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。如果找到返回值，返回出现处的下标；否则返回-1。

```
public int lastIndexOf(String str,int fromIndex)
```

入口参数

str：要搜索的子字符串。

fromIndex：开始反向搜索的索引位置。

返回值：指定子字符串在此字符串中最后一次出现处的索引。

示例

本示例使用 lastIndexOf 函数搜索字符串“编程词典”在字符串 strCom 中最后出现的位置，指定开始搜索的索引位置是 5，并将返回结果赋值给 int 型变量 index。由于执行的是反向搜索，所以从索引位置 5 开始进行反向搜索，返回结果将为-1。

```
String strCom = "明日科技，明日编程词典，明日软件";
int index = strCom.lastIndexOf("编程词典",5);
```

1.21 length 方法——获取字符串长度

该方法用于获取字符串的长度，并以 int 型变量返回该字符串的长度。

方法声明

```
public int length();
```

返回值：此对象表示的字符序列的长度。

示例

本示例使用 length 函数获取字符串 strCom1 的长度，并将返回的结果赋值给 int 型变量 length。

```
String strCom1 = "I like Java";
int length = strCom1.length();
```

示例

本示例使用 length 函数获取字符串 strCom2 的长度，并将返回的结果赋值给 int 型变量 length。

```
String strCom2 = "明日编程词典";
int length = strCom2.length();
```

1.22 matches 方法——匹配正则表达式

告知此字符串是否匹配给定的正则表达式。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public boolean matches(String regex)
```

入口参数：regex：用来匹配此字符串的正则表达式。

返回值：当且仅当此字符串匹配给定的正则表达式时，返回 true。

抛出异常

PatternSyntaxException：如果正则表达式的语法无效。

示例

本示例使用 matches 方法，判断字符串 regex 是否满足正则表达式\\d{4}，结果返回 true。

```
String regex = "1234";
System.out.println(regex.matches("\\d{4}"));
```

1.23 offsetByCodePoints 方法——获取索引偏移后指定代码点的索引

返回此 String 中从给定的 index 处偏移 codePointOffset 个代码点的索引。文本范围内由 index 和 codePointOffset 给定的未配对代理项各计为一个代码点。

方法声明

```
public int offsetByCodePoints(int index, int codePointOffset)
```

入口参数

index：要偏移的索引。

CodePointOffset：代码点中的偏移量。

返回值：此字符串索引。

示例

本示例使用 offsetByCodePoints 方法，返回字符串 strCom 从索引 2 开始偏移 3 个代码点的索引，输出为 5。

```
String strCom = "I LIKE JAVA";          // 定义一个字符串
int i = strCom.offsetByCodePoints(2,3); // 返回 strCom 索引
System.out.println(i);
```

1.24 regionMatches 方法——测试两个字符串区域是否相等

测试两个字符串区域是否相等。

方法声明 1

```
public boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
```

入口参数

ignoreCase：如果为 true，则比较字符时忽略大小写。

toffset：此字符串中子区域的起始偏移量。

other：字符串参数。

ooffset：字符串参数中子区域的起始偏移量。

len：要比较的字符数。

返回值：

如果此字符串的指定子区域完全匹配字符串参数的指定子区域，则返回 true；否则返回 false。

注意

将此 String 对象的子字符串与参数 other 的子字符串进行比较。如果这两个子字符串表示相同的字符序列，则结果为 true，当且仅当 ignoreCase 为 true 时忽略大小写。要比较的此 String 对象的子字符串从索引 toffset 处开始，长度为 len。要比较的 other 的子字符串从索引 ooffset 处开始，长度为 len。当且仅当下列至少一项为 true 时，结果才为 false:

toffset 为负；

ooffset 为负；

toffset+len 大于此 String 对象的长度；

ooffset+len 大于另一个参数的长度；

ignoreCase 为 false；

ignoreCase 为 true。

示例

本示例使用 regionMatches 方法，忽略大小写比较 strCom、subStr 从索引 2 开始比较 3 个字符串是否相等，返回布尔值。

```
String strCom = "ABLIKE";          // 定义一个字符串
String subStr = "CDlike";          // 定义一个字符串
//比较两个字符串
boolean b = subStr.regionMatches(true, 2, strCom, 2, 3);
System.out.println(b);
```

方法声明 2

```
public boolean regionMatches(int toffset, String other, int ooffset, int len)
```

入口参数

toffset：此字符串中子区域的起始偏移量。

other：字符串参数。

PDF电子书基地 http://dayo1982.400gb.com

ooffset：字符串参数中子区域的起始偏移量。

len：要比较的字符数。

返回值：如果此字符串的指定子区域完全匹配字符串参数的指定子区域，则返回 true；否则返回 false。

注意

将此 String 对象的一个子字符串与参数 other 的一个子字符串进行比较。如果这两个子字符串表示相同的字符序列，则结果为 true。要比较的此 String 对象的子字符串从索引 toffset 处开始，长度为 len。要比较的 other 的子字符串从索引 ooffset 处开始，长度为 len。当且仅当下列至少一项为 true 时，结果才为 false:

toffset 为负；

ooffset 为负；

toffset+len 大于此 String 对象的长度；

ooffset+len 大于另一个参数的长度；

存在某个小于 len 的非负整数 k，它满足：this.charAt(toffset+k)!=other.charAt(ooffset+k)。

示例

本示例使用 regionMatches 方法，比较 strCom、subStr，从索引 2 开始比较 3 个字符串是否相等，返回布尔值。

```
String strCom = "ABLIKE"; // 定义一个字符串
String subStr = "CDLIKE"; // 定义一个字符串
//比较两个字符串
boolean b = subStr.regionMatches(2, strCom, 2, 3);
System.out.println(b);
```

1.25 replace 方法——替换字符序列

应用该方法中的 newChar 替换指定字符串中出现的所有 oldChar，返回一个新的字符串。

方法声明 1

```
public String replace(char oldChar , char newChar)
```

入口参数

olderChar：要替换的子字符串或者字符。

newChar：新的字符串或字符，用于替换原有字符串的内容。

返回值：一个从此字符串派生的字符串，它将此字符串中的所有 oldChar 替代为 newChar。

示例

本示例使用 replace()方法将字符串“明日图书”中的内容，修改为“明日编程词典”。

```
String str="明日图书";
str=str.replace("明日图书", "明日编程词典");
```

方法声明 2

使用指定的字面值替换序列替换此字符串所有匹配字面值目标序列的子字符串。该替换从字符串的开头向末尾执行。

```
public String replace(CharSequence target,CharSequence replacement)
```

入口参数

target：要被替换的 char 值序列。

replacement：char 值的替换序列。

返回值：所得的 String。

示例

本示例使用 replace()方法用字符串“m”替换字符串“nnn”中的内容，将“nn”替换为“m”，并输出替换后的字符串。由于该方法是从字符串开头向末尾执行，所以返回结果是“mn”，而不是“nm”。

```
String str="nnn";
str=str.replace("nn", "m");
```

1.26　replaceAll 方法——正则表达式匹配替换所有字符串

使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串。

方法声明

```
public String replaceAll(String regex,String replacement)
```

入口参数

regex：用来匹配此字符串的正则表达式。

replacement：用来替换每个匹配项的字符串。

返回值：所得 String。

抛出异常

PatternSyntaxException：如果正则表达式的语法无效。

示例

本示例使用 replaceAll 函数将字符串“成功者找方法，”中的内容修改为“成功者找方法失败者找理由”。

PDF电子书基地 http://dayo1982.400gb.com

```
String str="成功者找方法 ， ";                    //定义字符串
str=str.replaceAll("， ","失败者找理由");          //替换字符串中的逗号
System.out.println(str);                          //输出返回的字符串
```

1.27 replaceFirst 方法——正则表达式替换第一个匹配的字符串

使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串。

方法声明

```
public String replaceFirst(String regex,String replacement)
```

入口参数

regex：用来匹配此字符串的正则表达式。

replacement：用来替换第一个匹配项的字符串。

返回值：所得 String。

示例

本示例使用 replaceFirst 函数替换字符串“明日编程词典，明日图书，明日软件”中的第一个“明日”，并将返回值赋给 String 变量 str。

```
public static void main(String[] args) {
    String str="明日编程词典，明日图书，明日软件";     //定义字符串
    str=str.replaceFirst("明日", "JAVA");              //替换字符串中第一个“明日”字符
    System.out.println(str);                           //输出结果
}
```

1.28 split 方法——字符串分割成数组

按指定的分割字符或字符串将字符串内容分割并存放到字符串数组中。

方法声明 1

```
public String[] split(String regex)
```

入口参数：regex 为指定的字符串分隔符，也可以使用正则表达式来分割字符串。

返回值：字符串数组，它是根据给定正则表达式的匹配拆分此字符串确定的。

抛出异常

PatternSyntaxException：如果正则表达式的语法无效。

示例

本示例使用 split()方法以“!”和“。”分隔符将字符串 str 中的公司信息分割成数组，并通过 for 循环语句输出字符串数组中的内容。

```
String str="公司名称：明日科技!公司所在城市：长春市。公司电话：84972266";
String[] info=null;
```

```
info=str.split("!|。");
for(int i=0;i<info.length;i++){                    //输出返回数组
        System.out.println(info[i]);
    }
```

示例

本示例使用 split()方法以“,”为分隔符，将字符串 str 中的数据转换成数组的形式，并通过 for 循环语句输出字符串数组中的内容。

```
String str="JAVA 程序设计自学手册，PHP 网络编程自学手册，ASP 网络编程自学手册";
String[] info=null;
info=str.split(", ");                              //以逗号为分隔符
for(int i=0;i<info.length;i++){                    //输出返回数组
    System.out.println(info[i]);
}
```

方法声明 2

```
public String[] split(String regex, int limit)
```

入口参数

regex：为指定的字符串分隔符，也可以使用正则表达式来分割字符串。

limit：控制模式应用的次数，因此影响所得数组的长度。

返回值：字符串数组，它是根据给定正则表达式的匹配拆分此字符串确定的。

抛出异常

PatternSyntaxException：如果正则表达式的语法无效。

注意

如果该限制 n 大于 0，则模式将被最多应用（n–1）次，数组的长度将不会大于 n，而且数组的最后一项将包含所有超出最后匹配的定界符的输入。如果 n 为非正，那么模式将被应用尽可能多的次数，而且数组可以是任何长度。如果 n 为 0，那么模式将被应用尽可能多的次数，数组可以是任何长度，并且结尾空字符串将被丢弃。

示例

本示例使用 split()方法以“,”为分隔符，将字符串 str 中的数据转换成数组的形式，这里将参数 limit 的值设置为 2，即执行两次匹配，并通过 for 循环语句输出字符串数组中的内容。

```
String str="JAVA 程序设计自学手册，PHP 网络编程自学手册，ASP 网络编程自学手册";
String[] info=null;
info=str.split(", ",2);
for(int i=0;i<info.length;i++){                    //输出返回数组
    System.out.println(info[i]);
}
```


PDF电子书基地 http://dayo1982.400gb.com

注意

本示例的返回结果不在是上一示例中的3个元素值，而是2个元素值。其中的第2元素值中包含第2次匹配后的所有内容。这就是参数limit的作用，控制匹配执行的次数。

1.29 startsWith 方法——判断前缀字符串

测试此字符串从指定索引开始的子字符串是否以指定前缀开始。

方法声明 1

```
public boolean startsWith(String prefix , int toffset)
```

参数说明

prefix：前缀。

toffset：在字符串中开始查找的位置。

返回值：如果参数表示的字符序列是此对象从索引 toffset 处开始的子字符串，则返回 true；否则返回 false。如果 toffset 为负或大于此 String 对象的长度，则结果为 false；否则结果与该表达式的结果相同：

```
this.substring(toffset).startsWith(prefix)。
```

示例

本示例使用 startsWith 函数来判断前缀“I l”是否是字符串 strCom1 中开始索引位置是 0 的字符序列，并将结果赋值给 boolean 变量 strB。由于字符串 strCom1 中开始索引位置在 0 的字符序列与指定的前缀“I l”相同，因此对象 strB 为 true。

```
String strCom1 = "I like Java";
boolean strB = strCom1.startsWith("I l",0);
```

方法声明 2

测试此字符串是否以指定的前缀开始。

```
public boolean startsWith(String prefix)
```

入口参数：prefix 为指定的前缀。

返回值：如果参数表示的字符序列是此字符串表示的字符序列的前缀，则返回 true；否则返回 false；如果参数是空字符串，或者等于此 String 对象（用 equals(Object)方法确定），则返回 true。

示例

本示例使用 startsWith 函数来判断字符串 strCom1，是否以字符串“like”开

始，并将结果赋值给 boolean 变量 strB。由于字符串 strCom1 不是以字符串“like”开始的，因此 boolean 类型变量 strB 的值为 false。

```
String strCom1 = "I like Java";
boolean strB = strCom1.startsWith("like");
```

1.30 subSequence 方法——获取子字符串序列

返回一个新的字符序列，它是此序列的一个子序列。

方法声明

```
public CharSequence subSequence(int beginIndex, int endIndex)
```

入口参数

beginIndex：起始索引（包括）。

endIndex：结束索引（不包括）。

返回值：此字符串的指定子序列。

抛出异常

IndexOutOfBoundsException：如果 beginIndex 或 endIndex 为负，如果 endIndex 大于 length()或 beginIndex 大于 startIndex。

示例

本示例使用 subSequence 方法，输出字符串索引 2～6 的子字符串，相当于 subString()方法。

```
String strCom = "I LIKE JAVA"; // 定义一个字符串
System.out.println(strCom.subSequence(2, 6));
```

1.31 substring 方法——获取子字符串

该方法实现对字符串从指定的索引位置开始截取，直到此字符串的末尾，并返回一个新的字符串。

方法声明 1

```
public String substring(int beginIndex)
```

入口参数：beginIndex 为开始处的索引（包括该位置）。

返回值：指定的子字符串。

示例

本示例使用 substring 函数截取字符串 strCom，索引位置从 3 开始截取到字符串 strCom 的结尾处，并将截取的结果赋值给 String 类型变量 strResult。

```
String strCom = "I Like Java";
String strResult = strCom.substring(3);
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

该方法实现对字符串从指定的索引位置 beginIndex 开始截取，直到索引位置为 endIndex-1 处结束，并返回一个长度为 endIndex-beginIndex 的新字符串。

```
public String substring(int beginIndex, int endIndex)
```

入口参数

beginIndex：开始处的索引（包括该位置）。

endIndex：结束索引（不包括该索引位置）。

返回值：指定的子字符串。

示例

本示例使用 beginIndex 函数截取字符串 strCom，从索引位置 3 处开始到索引位置 5 处结束，并将截取的字符串赋值给 String 类型变量 strResult。

```
String strCom = "I Like Java";
String strResult = strCom.substring(3,5);
```

1.32 toCharArray 方法——字符串变字符数组

将字符串转换成一个字符数组。返回值为一个新分配的字符数组，它的长度是此字符串的长度，它的内容被初始化为包含此字符串表示的字符序列。

方法声明

```
public char[] toCharArray()
```

返回值：一个新分配的字符数组，它的长度是此字符串的长度，它的内容被初始化为包含此字符串表示的字符序列。

示例

本示例使用 toCharArray 函数将字符串 strCom 转换成一个数组，并将转换后的结果赋值给 char[]变量 str。

```
public static void main(String[] args) {
        String strCom="JAVA 编程词典";          //定义字符串
    char[] str=strCom.toCharArray();           //将字符串转换成数组
    for(int i=0;i<str.length;i++){             //输出返回数组
        System.out.println(str[i]);
    }
}
```

1.33 toLowerCase 方法——转换成小写字符串

该方法用于将大写字母转换成小写字母，返回值为 String 类型。

方法声明 1

```
public String toLowerCase()
```

返回值：转换为小写的 String。

示例

本示例使用 toLowerCase 函数将字符串 strCom 中的大写字母转换成小写字母，并将转换后的结果赋值给 String 变量 strLower。

```
String strCom = "I LIKE JAVA";
String strLower = strCom.toLowerCase();
```

方法声明 2

```
public String toLowerCase(Locale locale)
```

入口参数：locale 为使用此语言环境的大小写转换规则。

返回值：转换为小写的 String。

示例

本示例使用 toLowerCase 方法将字符串 strCom 中的大写字母按指定的 Locale 规则转换成小写字母，并将转换后的结果赋值给 String 变量 strLower。

```
String strCom = "I LIKE JAVA";
String strLower = strCom.toLowerCase(Locale.UK);
```

1.34 toUpperCase 方法——转换成大写字符串

使用默认语言环境的规则，将字符串中的所有字符都转换成大写。该方法等效于 toUpperCase(Locale.getDefault())。返回值为 String 类型。

方法声明 1

```
public String toUpperCase()
```

返回值：转换为大写的 String。

示例

本示例使用 toUpperCase 函数将字符串 strCom1 中的所有小写字母转换为大写字母，并将返回的结果赋值给 String 型变量 strCom。

```
String strCom1 = "I like Java";
String strCom = strCom1.toUpperCase();
```

方法声明 2

```
public String toUpperCase(Locale locale)
```

返回值：转换为大写的 String。

示例

本示例使用 toUpperCase 方法将字符串 strCom1 中的所有小写字母按指定的 Locale 规则转换为大写字母，并将返回的结果赋值给 String 型变量 strCom。

PDF电子书基地 http://dayo1982.400gb.com

```
String strCom1 = "I like Java";
String strCom = strCom1.toUpperCase(Locale.US);
```

1.35 trim 方法——截取空白

返回字符串的副本，即去除字符串的前后空白。

方法声明

```
public String trim()
```

返回值：字符串移除了前导和尾部空白的副本；如果没有前导和尾部空白，则返回此字符串。

注意

如果此 String 对象表示一个空字符序列，或者此 String 对象表示的字符序列的第一个和最后一个字符的代码都大于'\u0020'(空格字符), 则返回对此 String 对象的引用。

否则，若字符串中没有代码大于'\u0020'的字符，则创建并返回一个表示空字符串的新 String 对象。

否则，假定 k 为字符串中代码大于'\u0020'的第一个字符的索引，m 为字符串中代码大于'\u0020'的最后一个字符的索引，创建一个新的 String 对象，它表示此字符串中从索引 k 处的字符开始，到索引 m 处的字符结束的子字符串，即 this.substring(k, m+1)的结果。

示例

本示例使用 trim 函数去除字符串中前后的空格，并同时输出去除空格前和去除空格后的字符串。

```
public static void main(String[] args) {
    String strCom="      JAVA 编程词典     ";          //定义字符串
    String str=strCom.trim();                          //去除字符串前后的空格
    System.out.println("未去除前后空格的字符串："+strCom);
    System.out.println("去除前后空格后的字符串："+str);
}
```

1.36 valueOf 方法——基本数据类型转成字符串

valueOf 是一个静态方法，返回指定类型参数的字符串表示形式。指定类型参数可以是：boolean、char、int、long、float 和 double。

方法声明 1

返回 boolean 参数的字符串表示形式。如果参数为 true，则返回一个等于"true"的字符串；否则，返回一个等于"false"的字符串。

```
public static String valueOf(boolean b)
```

入口参数：b 是一个 boolean。

返回值：一个新分配的字符串，它表示包含在字符数组参数中的相同字符序列。

示例

本示例使用 valueOf 函数获取 boolean 型数据的字符串形式。

```
public static void main(String[] args) {
    boolean strCom=true;
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

方法声明 2

返回 char 参数的字符串表示形式。一个长度为 1 的字符串，它包含参数 c 的单个字符。

```
public static String valueOf(char c)
```

入口参数：c 是一个 char。

返回值：一个长度为 1 的字符串，它包含参数 c 的单个字符。

示例

本示例使用 valueOf 函数获取 char 类型数据的字符串形式。

```
public static void main(String[] args) {
    char strCom='明';
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

方法声明 3

返回 int 参数的字符串表示形式。该表示形式恰好是单参数的 Integer.toString 方法返回的结果。

```
public static String valueOf(int i)
```

入口参数：I 是一个 int。

返回值：int 参数的字符串表示形式。

示例

本示例使用 valueOf 函数获取 int 类型数据的字符串形式。

```
public static void main(String[] args) {
    int strCom=2008;                          //定义 int 型数据
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

方法声明 4

返回 long 参数的字符串表示形式。该表示形式恰好是单参数的 Long.toString

PDF电子书基地 http://dayo1982.400gb.com

方法返回的结果。

```
public static String valueOf(long l)
```

入口参数：l 是一个 long。

返回值：long 参数的字符串表示形式。

示例

本示例使用 valueOf 函数获取 long 类型数据的字符串形式。

```
public static void main(String[] args) {
    long strCom=214L;                          //定义长整型数据
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

方法声明 5

返回 float 参数的字符串表示形式。该表示形式恰好是单参数的 Float.toString 方法返回的结果。

```
public static String valueOf(float f)
```

入口参数：f 是一个 float。

返回值：float 参数的字符串表示形式。

示例

本示例使用 valueOf 函数获取 float 类型数据的字符串形式。float 类型数据的值为十进制表示的单精度小数（在数字后加 F）。

```
public static void main(String[] args) {
    float strCom=12.12F;                       //定义浮点型数据
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

示例

本示例使用 valueOf 函数获取 float 类型数据的字符串形式。float 类型数据的值为科学计算法表示的单精度小数（在数字后加 F）。

```
public static void main(String[] args) {
    float strCom=1.212E1F;                     //定义浮点型数据
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

方法声明 6

返回 double 参数的字符串表示形式。该表示形式恰好是单参数的 Double.toString 方法返回的结果。

```
public static String valueOf(double d)
```

入口参数：d 是一个 double。

返回值：double 参数的字符串表示形式。

示例

本示例使用 valueOf 函数获取 double 类型数据的字符串形式。其中 double 类型数据的值为十进制表示的双精度小数。

```
public static void main(String[] args) {
    double strCom=56.78;                          //定义 double 型数据，十进制双精度小数
    String str=String.valueOf(strCom);
    System.out.println(str);
}
```

示例

本示例使用 valueOf 函数获取 double 类型数据的字符串形式。其中 double 类型数据的值为科学计算法表示的双精度小数。

```
public static void main(String[] args) {
    double strCom=567.8E-1;                       //定义 double 型数据，科学计算法表示的值
    String str=String.valueOf(strCom);
    System:out.println(str);
}
```

方法声明 7

返回 char 数组参数的字符串表示形式。字符数组的内容已被复制，后续修改不会影响新创建的字符串。

```
public static String valueOf(char[] data)
```

入口参数：data 是一个字符数组。

返回值：一个新分配的字符串，它表示包含在字符数组参数中的相同字符序列。

示例

本示例使用 valueOf 方法获取与数组序列相同的字符串形式。

```
char[] array={'明','日','科','技'};//定义一个字符数组
String data = String.valueOf(array);
System.out.println(data);
```

方法声明 8

返回 char 数组参数的特定子数组的字符串表示形式。

offset 参数是子数组的第一个字符的索引。count 参数指定子数组的长度。字符数组的内容已被复制，后续修改不会影响新创建的字符串。

```
public static String valueOf(char[] data, int offset, int count)
```

入口参数

data：字符数组。

offset：String 值的初始偏移量。

count：String 值的长度。

PDF电子书基地 http://dayo1982.400gb.com

返回值：一个字符串，它表示在字符数组参数的子数组中包含的字符序列。

抛出异常

IndexOutOfBoundsException：如果 offset 为负，count 为负，或者 offset+count 大于 data.length。

示例

本示例使用 valueOf 方法获取 char 数组初始偏移量 2、长度为 2 的字符串。

```
char[] array={'明','日','科','技'};                    //定义一个字符数组
String data = String.valueOf(array,2,2);
System.out.println(data);
```

方法声明 9

返回 Object 参数的字符串表示形式。

```
public static String valueOf(Object obj)
```

入口参数：obj 是一个对象。

返回值：如果参数为 null，则字符串等于"null"；否则，返回 obj.toString() 的值。

示例

本示例使用 valueOf 方法获取 Integer 对象的字符串形式。

```
Integer o = new Integer(123);                          //定义一个 Integer 对象
String str = String.valueOf(o);
System.out.println(str);
```

第2章 包 装 类

Java 是一种面向对象语言，Java 中的类把方法与数据连接在一起，构成了自包含式处理单元。但在 Java 中不能定义基本数据类型对象，为了能将基本数据类型视为对象来处理。Java 中为每个基本数据类型提供了包装类，这样，我们便可以把基本类型转化为对象来处理。

2.1 Integer 类——整数类

java.lang 包中的 Integer 类是基本数据类型 int 的包装类，该类包含一个 int 类型的字段，并提供了处理 int 类型时非常有用的常量和方法。

2.1.1 构造方法——Integer 类的构造方法

Integer 类的构造方法

构 造 方 法	使 用 说 明
Integer(int value)	构造一个 Integer 对象，表示指定的 int 值
Integer(String str)	构造一个 Integer 对象，表示 String 参数所指定的 int 值

2.1.2 常量——Integer 类的常量

Integer 类的常量

常 量	含 有	数 值
MAX_VALUE	表示 int 类型能够表示的最大值	$2^{31}-1$
MIN_VALUE	表示 int 类型能够表示的最小值	-2^{31}
SIZE	用来以二进制补码表示 int 值的比特位数	—
TYPE	表示基本类型 int 的 Class 实例	—

2.1.3 bitCount 方法——获取二进制补码中 1 的数量

返回指定 int 值的二进制补码表示形式的 1 位的数量。

方法声明

```
public static int bitCount(int i)
```

入口参数：i 为 int 值。

返回值：返回指定 int 值的二进制补码表示形式的 1 位的数量。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例通过 bitCount()方法分别输出了 0～9 二进制补码表示形式的 1 位的数量。

```
for (int i = 0; i < 10; i++) {
    int count = Integer.bitCount(i);
    System.out.println(count);
}
```

2.1.4 byteValue 方法——获取 byte 类型的值

以 byte 类型返回该 Integer 的值。

方法声明

```
public byte byteValue();
```

返回值：转换为 byte 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 byteValue()方法将 Integer 对象转换为 byte 变量，并将转换后的结果输出。

```
Integer bytes = new Integer("125");
byte str = bytes.byteValue();
```

2.1.5 compareTo 方法——比较整数

在数字上比较两个 Integer 对象。

方法声明

```
public int compareTo(Integer anotherInteger);
```

入口参数：anotherInteger 是要比较的 Integer。

返回值：如果该 Integer 等于 Integer 参数，则返回 0 值；如果该 Integer 在数字上小于 Integer 参数，则返回小于 0 的值；如果 Integer 在数字上大于 Integer 参数，则返回大于 0 的值（有符号的比较）。

示例

本示例创建两个 Integer 对象，应用 compareTo()方法比较 Integer 对象的大小，将比较结果赋给 int 型变量 str。

```
Integer bytes = new Integer("125");
Integer bytes1 = new Integer("125");
int str = bytes.compareTo(bytes1);
```

2.1.6 decode 方法——字符串解码为 int 类型

将 String 解码为 Integer。

方法声明

```
public static Integer decode(String nm)
        throws NumberFormatException;
```

入口参数：nm 指定要解码的 string。

返回值：返回 nm 所表示的 int 值的 Integer 对象。

抛出异常

NumberFormatException：如果 String 不包含可解析整数。

示例

本示例创建 String 对象，并通过 decode()方法将 String 对象的参数进行解码，并将返回结果赋给 int 型变量 str。

```
String stri =new String("100");
int str=Integer.decode(stri);
```

2.1.7 doubleValue 方法——返回 double 数值

以 double 类型返回该 Integer 的值。

方法声明

```
public double doubleValue();
```

返回值：转换为 double 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 doubleValue()方法将 Integer 对象转换为 double 变量，并将转换后的结果赋给 double 变量 str。

```
Integer doubles = new Integer(123);
double str = doubles.doubleValue();
```

2.1.8 equals 方法——判断整数对象相等

比较此对象与指定对象。当且仅当参数不为 null，并且是一个与该对象包含相同 int 值的 Integer 对象时，结果为 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

示例

本示例创建两个 Integer 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。

PDF电子书基地 http://dayo1982.400gb.com

```
Integer i = new Integer(123);
Integer j = new Integer(123);
boolean b = i.equals(j);
System.out.println(b);
```

示例

本示例创建一个 Integer 对象、一个 String 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。只有比较包含相同 int 值的 Integer 对象时，结果为 true。

```
Integer i = new Integer(123);
String s = "123";
boolean b = i.equals(s);
System.out.println(b);
b = i.equals(Integer.valueOf(s));
System.out.println(b);
```

2.1.9 floatValue 方法——获取 float 数值

以 float 类型返回该 Integer 的值。

方法声明

```
public float floatValue();
```

返回值：转换为 float 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 floatValue()方法将 Integer 对象转换为 float 变量，并将转换后的结果赋给 float 变量 str。

```
Integer floats = new Integer(123);
float str = floats.floatValue();
```

2.1.10 getInteger 方法——获取整数的系统属性值

确定具有指定名称的系统属性的整数值。

方法声明 1

```
public static Integer getInteger(String nm)
```

入口参数：nm 为属性名。

返回值：此方法返回属性的 Integer 值。

注意

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问系统属性。然后，将该属性的字符串值解释为一个整数值，并返回表示该值的 Integer 对象。使用 getProperty 的定义可以找到可能出现的数字格式的详细信息。

如果没有具有指定名称的属性，或者指定名称为空或 null，或者属性的数字格式不正确，则返回 null。

示例

本示例通过 getInteger()方法返回了系统属性名称为"sun.arch.data.model"整数值。

```
String nm = "sun.arch.data.model";
Integer i = Integer.getInteger(nm);
System.out.println(i);
```

方法声明 2

确定具有指定名称的系统属性的整数值，第二个参数是默认值。

```
public static Integer getInteger(String nm, int val)
```

入口参数

nm：属性名。

val：int 型默认值。

返回值：此方法返回属性的 Integer 值。

注意

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问系统属性。然后，将该属性的字符串值解释为一个整数值，并返回表示该值的 Integer 对象。使用 getProperty 的定义可以找到可能出现的数字格式的详细信息。

第二个参数是默认值。如果未具有指定名称的属性，或者属性的数字格式不正确，或者指定名称为空或 null，则返回一个表示第二个参数的值的 Integer 对象。

示例

本示例通过 getInteger()方法返回了系统属性名称为"java.version"整数值，由于返回值为 null，所以返回了默认值为 10。

```
String nm = "java.version";
Integer i = Integer.getInteger(nm,10);
System.out.println(i);
```

方法声明 3

确定具有指定名称的系统属性的整数值。第二个参数是默认值。

```
public static Integer getInteger(String nm, Integer val)
```

入口参数

nm：属性名。

val：Integer 型默认值。

返回值：此方法返回属性的 Integer 值。

PDF电子书基地 http://dayo1982.400gb.com

注意

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问系统属性。然后，根据每个 Integer.decode 方法，将该属性的字符串值解释为一个整数值，并返回一个表示该值的 Integer 对象。如果属性值以两个 ASCII 字符 0x 或者 ASCII 字符#开始，并且后面没有减号，则将它的剩余部分解析为十六进制整数，就好像以 16 为基数调用 valueOf(java.lang.String,int)方法一样。

如果属性值以 ASCII 字符 0 开始，后面还有其他字符，则将它解析为八进制整数，就好像以 8 为基数调用 valueOf(java.lang.String,int)方法一样。否则，将属性值解析为十进制整数，就好像以 10 为基数调用 valueOf(java.lang.String, int)方法一样。

第二个参数是默认值。如果未具有指定名称的属性，或者属性的数字格式不正确，或者指定名称为空或 null，则返回默认值。

示例

本示例创建 1 个 Integer 对象，通过 getInteger()方法返回了系统属性名称为“java.version”整数值，由于返回值为 null，所以返回了默认值为 10。

```
String nm = "java.version";
Integer j = new Integer(10);
Integer i = Integer.getInteger(nm,j);
System.out.println(i);
```

2.1.11 hashCode 方法——生成整数的哈希码

此方法返回此 Integer 的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回此 Integer 的哈希码。

示例

本示例创建 1 个 Integer 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Integer i = new Integer(10);
int j = i.hashCode();
System.out.println(j);
```

2.1.12 highestOneBit 方法——获取整数二进制最高位 1 的索引

此方法返回具有至多单个 1 位的 int 值，在指定的 int 值中最高位（最左边）的 1 位的位置。如果指定的值在其二进制补码表示形式中不具有 1 位，即它等于零，则返回零。

方法声明

```
public static int highestOneBit(int i)
```

入口参数：i 为 int 值。

返回值：返回具有单个 1 位的 int 值，在指定值中最高位的 1 位的位置，否则，如果指定值本身等于零，则返回零。

示例

本示例通过 highestOneBit()方法依次输出了 0～9 中最高位的 1 位的位置，具有单个 1 位的 int 值。

```
for (int i = 0; i < 10; i++) {
        int j = Integer.highestOneBit(i);
        System.out.println(j);
}
```

2.1.13 intValue()方法——获取 int 值

该方法以 int 类型返回该 Integer 的值。

方法声明

```
public int intValue();
```

返回值：转换为 int 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 intValue()方法将 Integer 对象转换为 int 变量，并将转换后的结果输出。

```
Integer ints = new Integer("12");
int str = ints.intValue();
```

2.1.14 longValue 方法——获取 long 值

以 long 类型返回该 Integer 的值。

方法声明

```
public long longValue();
```

返回值：转换为 long 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 longValue()方法将 Integer 对象转换为 long 变量，并将转换后的结果输出。

```
Integer longs = new Integer("12");
long str = longs.longValue();
```


PDF电子书基地 http://dayo1982.400gb.com

2.1.15 lowestOneBit 方法——获取整数二进制最低位 1 的索引

此方法返回具有至多单个 1 位的 int 值，在指定的 int 值中最低位（最右边）的 1 位的位置。如果指定的值在其二进制补码表示形式中不具有 1 位，即它等于零，则返回零。

方法声明

```
public static int lowestOneBit(int i)
```

入口参数：i 为 int 值。

返回值：返回具有单个 1 位的 int 值,在指定值中最低位的 1 位的位置，否则，如果指定值本身等于零，则返回零。

示例

本示例通过 lowestOneBit()方法依次输出了 0～9 中最低位（最右边）的 1 位的位置具有至多单个 1 位的 int 值。

```
for (int i = 0; i < 10; i++) {
        int j = Integer. lowestOneBit (i);
        System.out.println(j);
}
```

2.1.16 parseInt 方法——将字符串解析为 int 值

该方法将字符串参数作为有符号的十进制整数进行解析。需要注意的是字符串中的字符都必须是十进制数字。

方法声明 1

```
public static int parseInt(String s) throws NumberFormatException
```

入口参数：s 包含要解析的 int 表示形式的 String。

返回值：返回值为十进制的整数值。

抛出异常

NumberFormatException：如果 String 不包含可解析的 int。

示例

本示例创建 String 对象 stri，并通过 Integer 类的 parseInt()方法将 String 类型转换为 int 对象，并将转化后的结果输出。

```
String stri = new String("100");
int str = Integer.parseInt(stri);
```

示例

本示例创建 Main 类，定义主方法 main，在主方法中定义 String 数组，实现将 Strring 类型数组中的元素转换为 int 类型，并将各元素相加。

```
public class Main{                                          //创建类
public static void main(String[] args) {                    //定义主方法
    String str[] = {"15","20","45","35"};                   //定义 String 数组
    int sum = 0;                                            //定义 int 型变量 sum
    for(int i=0; i<str.length;i++){                         //循环遍历数组
        int myint =Integer.parseInt(str[i]);                //数组中每个元素都转换为 int 型
        sum =sum+=myint;                                    //将数组中的各元素相加
    }
    System.out.println("数组中各元素的和是："+sum);      //输出结果
}
}
```

方法声明 2

使用第二个参数指定的基数，将字符串参数解析为有符号的整数。除了第一个字符可以是用来表示负值的 ASCII 减号'-'('\u002D')外，字符串中的字符必须都是指定基数的数字（通过 Character.digit(char,int)是否返回一个负值确定）。返回得到的整数值。

```
public static int parseInt(String s, int radix)
    throws NumberFormatException
```

入口参数

s：包含要解析的 int 表示形式的 String。

radix：解析 s 时使用的基数。

返回值：使用指定基数的字符串参数表示的整数。

抛出异常

NumberFormatException：如果 String 不包含可解析的 int。

注意

如果发生以下任意一种情况，则抛出一个 NumberFormatException 类型的异常。

- 第一个参数为 null 或一个长度为零的字符串。
- 基数小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX。
- 假如字符串的长度超过 1，那么除了第一个字符可以是减号'-'('u002D')外，字符串中存在任意不是由指定基数的数字表示的字符。
- 字符串表示的值不是 int 类型的值。

示例

本示例创建 3 个 String 对象，并通过 Integer 类的 parseInt()方法将 String 类型转换为 int 对象，并将转化后的结果输出。

```
String str1 = new String("0");
String str2 = new String("123");
```


PDF电子书基地 http://dayo1982.400gb.com

```
String str3 = new String("-0");
int strResult1 = Integer.parseInt(str1,10);
int strResult2 = Integer.parseInt(str2,10);
int strResult3 = Integer.parseInt(str3,10);
```

示例

本示例创建 2 个 String 对象，并通过 Integer 类的 parseInt()方法将 String 类型转换为 int 对象，并将转化后的结果输出。

```
String str1 = new String("2147483647");
String str2 = new String("-2147483648");
int strResult1 = Integer.parseInt(str1,10);
int strResult2 = Integer.parseInt(str2,10);
```

示例

本示例创建 2 个 String 对象，并通过 Integer 类的 parseInt()方法将 String 类型转换为 int 对象。由于字符串中不包含可解析的 int，所以抛出异常。

```
String str3 = new String("2147483648");
String str4 = new String("99");
String str5 = new String("kona");
int strResult3 = Integer.parseInt(str3,10);
int strResult4 = Integer.parseInt(str4,8);
int strResult5 = Integer.parseInt(str5,10);
```

示例

本示例创建 3 个 String 对象，并通过 Integer 类的 parseInt()方法将 String 类型转换为 int 对象，并将返回值赋给 int 型变量。

```
String str1 = new String("-FF");
String str2 = new String("1100110");
String str3 = new String("kona");
int strResult1 = Integer.parseInt(str1,16);
int strResult2 = Integer.parseInt(str2,2);
int strResult3 = Integer.parseInt(str3,27);
```

2.1.17 reverse 方法——反转整数二进制补码的位顺序

返回通过反转指定 int 值的二进制补码表示形式中位的顺序而获得的值。

方法声明

```
public static int reverse(int i)
```

入口参数：i 为要反转指定的整数。

返回值：返回通过反转指定 int 值中位的顺序而获得的值。

示例

本示例通过 reverse 方法，反转了 10 的二进制补码表示形式中位的顺序。

```
int i;
i = Integer.reverse(10);
```

2.1.18 reverseBytes 方法——反转整数字节的顺序

返回通过反转指定 int 值的二进制补码表示形式中字节的顺序而获得的值。

方法声明

```
public static int reverseBytes(int i)
```

入口参数：i 为要反转指定的整数。

返回值：返回通过反转指定 int 值的字节而获得的值。

示例

本示例通过 reverseBytes 方法，反转了 10 的二进制补码表示形式中字节的顺序。

```
int i;
i = Integer. reverseBytes (10);
```

2.1.19 shortValue 方法——获取 short 值

以 short 类型返回该 Integer 的值。

方法声明

```
public short shortValue();
```

返回值：转换为 short 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 shortValue()方法将 Integer 对象转换为 int 变量，并将转换后的结果输出。

```
Integer shorts = new Integer("12");
short str = shorts.shortValue();
```

2.1.20 signum 方法——获取整数符号

返回指定 int 值的符号函数（如果指定值为负，则返回–1；如果指定值为零，则返回 0；如果指定的值为正，则返回 1）。

方法声明

```
public static int signum(int i)
```

入口参数：i 为要返回符号函数的整数。

返回值：返回指定 int 值的符号函数。

示例

本示例定义了 3 个整数，分别为正数、负数和 0，通过 signum()方法输出了

PDF电子书基地 http://dayo1982.400gb.com

它们的符号函数为：1、-1、0。

```
int i=10,j=-10,k=0;
System.out.println(Integer.signum(i));
System.out.println(Integer.signum(j));
System.out.println(Integer.signum(k));
```

2.1.21 toBinaryString 方法——生成整数的二进制字符串

以二进制（基数 2）无符号整数形式返回一个整数参数的字符串表示形式。

方法声明

```
public static String toBinaryString(int i);
```

入口参数：i 为要转换成字符串的整数。

返回值：用二进制（基数 2）参数表示的无符号整数值的字符串表示形式。

注意

如果参数为负，该无符号整数值为参数加上 2^{32}；否则等于该参数。将该值转换为二进制（基数 2）形式的无前导 0 的 ASCII 数字字符串。如果无符号数的大小为零，则用一个零字符'0'('\u0030')表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。字符'0'('\u0030')和'1'('\u0031')被用作二进制数字。

示例

本示例创建 Integer 对象，并通过 toBinaryString()方法获取到整数 10 的二进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toBinaryString(stri);
```

示例

本示例创建 Integer 对象，并通过 toBinaryString()方法获取到整数-10 的二进制无符号整数形式。

```
int stri =new Integer(-10);
String str=Integer.toBinaryString(stri);
```

2.1.22 toHexString 方法——生成整数的十六进制字符串

以十六进制（基数 16）无符号整数形式返回一个整数参数的字符串表示形式。

方法声明

```
public static String toHexString(int i);
```

入口参数：i 为要转换成字符串的整数。

返回值：参数的十六进制（基数 16）无符号整数值的字符串表示形式。

注意

如果参数为负，那么无符号整数值为参数加上 2^{32}；否则等于该参数。将该值转换为十六进制（基数 16）的无前导 0 的 ASCII 数字字符串。如果无符号数的大小值为零，则用一个零字符'0'('\u0030')表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。

示例

本示例创建 Integer 对象，并通过 toHexString()方法获取到整数 10 的十六进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toHexString(stri);
```

示例

本示例创建 Integer 对象，并通过 toHexString()方法获取到整数-10 的十六进制无符号整数形式。

```
int stri =new Integer(-10);
String str=Integer.toHexString(stri);
```

2.1.23 toOctalString 方法——生成整数的八进制字符串

以八进制（基数 8）无符号整数形式返回一个整数参数的字符串表示形式。

方法声明

```
public static String toOctalString(int i);
```

入口参数：i 为要转换成字符串的整数。

返回值：八进制参数（基数 8）表示的无符号整数值的字符串表示形式。

注意

如果参数为负，该无符号整数值为参数加上 2^{32}；否则等于该参数。该值被转换成八进制（基数 8）ASCII 数字的字符串，且没有附加前导 0。

如果无符号数大小为零，则用一个零字符'0'('\u0030')表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。用 01234567 作为八进制数字，它们是从'\u0030'到'\u0037'的字符。

示例

本示例创建 Integer 对象，并通过 toOctalString()方法获取到整数 10 的八进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toOctalString(stri);
```


PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 Integer 对象，并通过 toOctalString()方法获取到整数−10 的八进制无符号整数形式。

```
int stri =new Integer(−10);
String str=Integer.toOctalString(stri);
```

2.1.24 toString 方法——生成整数的十进制字符串

返回一个表示指定整数的 String 对象。将该参数转换为有符号的十进制表示形式，以字符串形式返回它，就好像将参数和基数 10 作为参数赋予 toString(int,int)方法。

方法声明 1

```
public static String toString(int i);
```

入口参数：i 为要转换成字符串的整数。

返回值：十进制（基数 10）参数的字符串表示形式。

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的十进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toString(stri);
System.out.println(str);
```

方法声明 2

返回用第二个参数指定基数表示的第一个参数的字符串表示形式。

```
public static String toString(int i, int radix);
```

入口参数

i：要转换成字符串的整数。

radix：用于字符串表示形式的基数。

返回值：使用指定基数的参数的字符串表示形式。

注意

如果基数小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX，则改用基数 10。

如果第一个参数为负，则结果中的第一个元素为 ASCII 的减号'−'('\u002D')。如果第一个参数为非负，则没有符号字符出现在结果中。

结果中的剩余字符表示第一个参数的大小。如果大小为零，则用一个零字符'0'('\u0030')表示；否则，大小的表示形式中的第一个字符将不是零字符。用以下

ASCII 字符作为数字：0123456789abcdefghijklmnopqrstuvwxyz，其范围是从'\u0030'到'\u0039'和从'\u0061'到'\u007A'。

如果 radix 为 N，则按照所示顺序，使用这些字符中的前 N 个作为其数字。因此，十六进制（基数为 16）的数字是 0123456789abcdef。如果希望得到大写字母，则可以在结果上调用 String.toUpperCase()方法：Integer.toString(n,16).toUpperCase()。

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的二进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toString(stri,2);
```

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的八进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toString(stri,8);
```

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的十进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toString(stri,10);
```

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的十六进制无符号整数形式。

```
int stri =new Integer(10);
String str=Integer.toString(stri,16);
```

方法声明 3

```
public String toString()
```

返回值：该对象的值（基数 10）的字符串表示形式。

示例

本示例创建 Integer 对象，并通过 toString()方法获取到整数 10 的十进制无符号整数形式。

```
int stri =new Integer(10);
String str= stri.toString();
System.out.println(str);
```


PDF电子书基地 http://dayo1982.400gb.com

2.1.25 valueOf 方法——创建 Integer 对象

返回一个表示指定的 int 值的 Integer 实例。如果不需要新的 Integer 实例，则通常应优先使用该方法，而不是构造方法 Integer(int)，因为该方法有可能通过缓存经常请求的值而显著提高空间和时间性能。

方法声明 1

```
public static Integer valueOf(int i);
```

入口参数：i 为一个 int 值。

返回值：表示 i 的 Integer 实例。

示例

本示例创建 Integer 对象，并通过 valueOf()方法获取 int 值 10 的 Integer 实例。

```
int stri =new Integer(10);
int str=Integer.valueOf(stri);
```

方法声明 2

返回保存指定的 String 的值的 Integer 对象。将该参数解释为表示一个有符号的十进制整数，就好像将该参数赋予 parseInt(java.lang.String)方法一样。结果是一个表示字符串指定的整数值的 Integer 对象。换句话说，该方法返回一个等于以下值的 Integer 对象：new Integer(Integer.parseInt(s))。

```
public static Integer valueOf(String s)
        throws NumberFormatException;
```

入口参数：s 为要解析的字符串。

返回值：保存字符串参数表示的值的 Integer 对象。

抛出异常

NumberFormatException：如果字符串不能解析为一个整数。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 10 的 Integer 对象。

```
String stri =new String("10");
int str=Integer.valueOf(stri);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值“明日科技”的 Integer 对象，由于该值不能解析为整数，所以抛出异常。

```
String stri =new String("明日科技");
int str=Integer.valueOf(stri);
System.out.println(str);
```

方法声明 3

返回一个 Integer 对象，该对象中保存了用第二个参数提供的基数进行解析时从指定的 String 中提取的值。

```
public static Integer valueOf(String s, int radix)
        throws NumberFormatException;
```

入口参数：

s：要解析的字符串。

radix：解释 s 时使用的基数。

返回值：一个 Integer 对象，它含有字符串参数（以指定的基数）所表示的数值。

抛出异常

NumberFormatException：如果字符串不能解析为一个整数。

注意

该方法将第一个参数解释为用第二个参数指定的基数表示的有符号整数，就好像将这些参数赋予 parseInt(java.lang.String,int)方法一样。结果是一个表示字符串指定的整数值的 Integer 对象。换句话说，该方法返回一个等于以下值的 Integer 对象：new Integer(Integer.parseInt(s,radix))。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 1010 以 2 为基数所表示的数值。

```
String stri =new String("1010");
int str=Integer.valueOf(stri,2);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 0123 以 8 为基数所表示的数值。

```
String stri =new String("0123");
int str=Integer.valueOf(stri,8);
```

2.2 Long 类——长整型类

java.lang 包中的 Long 类在对象中包装了基本类型 Long 的值。每个 Long 类型的对象都包含一个 Long 类型的字段。该类提供了多个方法，可以将 Long 转换为 String，将 String 转换为 Long。除此之外，还提供了其他一些处理 Long 时有用的常量和方法。

PDF电子书基地 http://dayo1982.400gb.com

2.2.1 构造方法——Long 类的构造方法

Long 类的构造方法

构 造 方 法	使 用 说 明
Long(long value)	构造一个 Long 对象，表示指定的 Long 参数
Long(String str)	构造一个 Long 对象，表示由 String 参数指示的 Long 值

2.2.2 常量——Long 类的常量

Long 类的常量

常　量	含　有	数　值
MAX_VALUE	表示 Long 类型的最大值	$2^{63}-1$
MIN_VALUE	表示 Long 类型的最小值	-2^{63}
SIZE	用来以二进制补码表示 Long 值的比特位数	
TYPE	表示基本类型 Long 的 Class 实例	

2.2.3 bitCount 方法——获取二进制补码中 1 的数量

返回指定 Long 值的二进制补码表示形式中的 1 位的数量。此功能有时被称为填充计算。

方法声明

```
public static int bitCount(long i)
```

入口参数：i 为 int 值。

返回值：返回指定 Long 值的二进制补码表示形式的 1 位的数量。

示例

本示例通过 bitCount()方法分别输出了 0～9 二进制补码表示形式的 1 位的数量。

```
for (int i = 0; i < 10; i++) {
    int count = Long.bitCount(i);
    System.out.println(count);
}
```

2.2.4 byteValue 方法——获取 byte 值

以 byte 类型返回该 Long 的值。

方法声明

```
public byte byteValue();
```

返回值：转换为 byte 类型后该对象表示的数值。

示例

本示例创建 Long 对象，并通过 byteValue()方法将 Long 对象转换为 byte 变量，并将转换后的结果输出。

```
Long bytes = new Long("125");
byte str = bytes.byteValue();
```

2.2.5 compareTo 方法——比较长整数

在数字上比较两个 Long 对象。

方法声明

```
public int compareTo(Long anotherLong);
```

入口参数：anotherLong 是要比较的 Long。

返回值：如果该 Long 等于 Long 参数，则返回 0 值；如果 Long 在数字上小于 Long 参数，则返回小于 0 的值；如果 Long 在数字上大于 Long 参数，则返回大于 0 的值（有符号的比较）。

示例

本示例创建两个 Long 对象，应用 compareTo()方法比较 Long 对象的大小，将比较结果赋给 int 型变量 str。

```
Long bytes = new Long("125");
Long bytes1 = new Long("125");
int str = bytes.compareTo(bytes1);
```

2.2.6 decode 方法——字符串解码为 long 类型

将 String 解码为 Long。

方法声明

```
public static Long decode(String nm)
        throws NumberFormatException;
```

入口参数：nm 指定要解码的 string。

返回值：nm 所表示的 Long 值的 Long 对象。

抛出异常

NumberFormatException：如果 String 不包含可解析整数。

示例

本示例创建 String 对象，并通过 decode()方法将 String 对象的参数进行解码，并将返回结果赋给 Long 型变量 str。

```
String stri =new String("100");
long str=Long.decode(stri);
```


PDF电子书基地 http://dayo1982.400gb.com

2.2.7 doubleValue 方法——返回 double 数值

以 double 类型返回该 Long 的值。

方法声明

```
public double doubleValue();
```

返回值：转换为 double 类型后该对象表示的数值。

示例

本示例创建 Long 对象，并通过 doubleValue()方法将 Long 对象转换为 double 变量，并将转换后的结果赋给 double 变量 str。

```
Long doubles = new Long(123);
double str = doubles.doubleValue();
```

2.2.8 equals 方法——判断长整数对象相等

将此对象与指定对象进行比较。当且仅当该参数不是 null，且 Long 对象与此对象包含相同的 long 值时，结果才为 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

示例

本示例创建两个 Long 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。

```
Long i = new Long(123);
Long j = new Long(123);
boolean b = i.equals(j);
System.out.println(b);
```

示例

本示例创建一个 Long 对象、一个 String 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。且 Long 对象与此对象包含相同的 long 值时，结果才为 true。

```
Long l = new Long(123);
String s = "123";
boolean b = l.equals(s);
System.out.println(b);
b = l.equals(Long.valueOf(s));
System.out.println(b);
```

2.2.9 floatValue 方法——获取 float 数值

以 float 类型返回该 Long 的值。

方法声明

```
public float floatValue();
```

返回值：转换为 float 类型后该对象表示的数值。

示例

本示例创建 Long 对象，并通过 floatValue()方法将 Long 对象转换为 float 变量，并将转换后的结果赋给 float 型变量 str。

```
Long floats = new Long(123);
float str = floats.floatValue();
```

2.2.10 getLong 方法——获取长整数的系统属性值

确定具有指定名称的系统属性的 Long 值。

方法声明 1

```
public static Long getLong(String nm)
```

入口参数：nm 为属性名。

返回值：此方法返回属性的 Long 值。

注意

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问该系统属性。然后，以 Long 值的形式解释此属性的字符串值，并返回表示此值的 Long 对象。在 getProperty 的定义中可以找到可能的数字格式的详细信息。

如果指定名称没有属性，或者指定名称为空或 null，或属性不具有正确的数字格式时，则返回 null。

示例

本示例通过 getLong()方法返回了系统属性名称为“sun.arch.data.model”整数值。

```
String nm = "sun.arch.data.model";
Long i = Long.getLong(nm);
System.out.println(i);
```

方法声明 2

确定具有指定名称的系统属性的 Long 值。第二个参数是默认值。

```
public static Long getLong(String nm, long val)
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：

nm：属性名。

Val：Long 型默认值。

返回值：此方法返回属性的 Long 值。

注意

使用指定名称确定系统属性的 Long 值。第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问该系统属性。然后，以 Long 值的形式解释此属性的字符串值，并返回表示此值的 Long 对象。在 getProperty 的定义中可以找到可能的数字格式的详细信息。

第二个参数是默认值。如果指定的名称没有属性，或者该属性不具备正确的数字格式，或指定名称为空或 null，则返回表示第二个参数的值的 Long 对象。

但是实际上，它可能通过以下方式实现：

```
Long result = getLong(nm,null);
return(result == null)?new Long(val):result;
```

这样可以避免不需要默认值时进行不必要的 Long 对象分配。

示例

本示例通过 getLong()方法返回了系统属性名称为“java.version”整数值，由于返回值为 null，所以返回了默认值为 10。

```
String nm = "java.version";
Long i = Long.getLong(nm,10);
System.out.println(i);
```

方法声明 3

确定具有指定名称的系统属性的 Long 值。第二个参数是默认值。

```
public static Long getLong(String nm, Long val)
```

入口参数

nm：属性名。

Val：Integer 型默认值。

返回值：此方法返回属性的 Integer 值。

注意

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String)方法可以访问该系统属性。然后，以 Long 值的形式解释此属性的字符串值，并且按照 Long.decode 方法返回表示此值的 Long 对象。如果该属性值以两个 ASCII 字符 0x 或 ASCII 字符#开头，后面没有跟减号，则将该属性的其余部分解析为一

个十六进制整数，该值与调用参数 radix 为 16 的 valueOf(java.lang.String,int)方法得到的值非常相似。

如果该属性值以 ASCII 字符 0 开头，后跟别的字符，则将它解析为一个八进制整数，该值与调用参数 radix 为 8 的 valueOf(java.lang.String,int)方法得到的值非常相似。否则，将属性值解析为一个十进制整数，该值与调用参数 radix 为 10 的 valueOf(java.lang.String,int)方法得到的值非常相似。

在所有情况下，都不允许将 L('\u004C')和 l('\u006C')作为类型指示符出现在属性值的结尾处，这一点在 Java 编程语言源代码中是允许的。

第二个参数是默认值。如果指定的名称没有属性，或者属性不具有正确的数字格式，或指定名称为空或 null，则返回默认值。

示例

本示例创建一个 Integer 对象，通过 getInteger()方法返回了系统属性名称为“java.version”整数值，由于返回值为 null，所以返回了默认值为 10。

```
String nm = "java.version";
Long j = new Long(10);
Long i = Long.getLong(nm,j);
System.out.println(i);
```

2.2.11 hashCode 方法——生成长整数的哈希码

此方法返回长整数的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回长整数的哈希码。

示例

本示例创建一个 Long 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Long i = new Long(10);
int j = i.hashCode();
System.out.println(j);
```

2.2.12 highestOneBit 方法——获取长整数二进制最高位 1 的索引

此方法返回返回至多有一个 1 位的 long 值，在指定的 long 值中最高位（最左边）的 1 位的位置。如果指定值在其二进制补码表示形式中没有 1 位，即等于零，则返回零。

方法声明

```
public static long highestOneBit(long i)
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：i 为 int 值。

返回值：返回具有单个 1 位的 long 值，在指定值中最高位的 1 位的位置；否则，如果指定值本身等于零，则返回零。

示例

本示例通过 highestOneBit()方法输出了 10 最高位的 1 位（最左边）的位置，具有单个 1 位的 long 值。

```
Long l = new Long(10);
long j = Long.highestOneBit(l);
System.out.println(j);
```

2.2.13 intValue()方法——获取 int 值

该方法以 int 类型返回该 Long 的值。

方法声明

```
public int intValue();
```

返回值：转换为 int 类型后该对象表示的数值。

示例

本示例创建 Long 对象，并通过 intValue()方法将 Long 对象转换为 int 变量，并将转换后的结果输出。

```
Long ints = new Long(22456L);
int str = ints.intValue();
```

2.2.14 longValue 方法——获取 long 值

以 long 值的形式返回此 Long 的值。

方法声明

```
public long longValue()
```

返回值：转换为 long 类型后该对象表示的数值。

示例

本示例创建 Integer 对象，并通过 longValue()方法将 Integer 对象转换为 long 变量，并将转换后的结果输出。

```
Long longs = new Long("12");
long str = longs.longValue();
```

2.2.15 lowestOneBit 方法——获取长整数二进制最低位 1 的索引

返回至多有一个 1 位的 long 值，在指定的 long 值中最低位（最右边）的 1 位的位置。如果指定值在其二进制补码表示形式中没有 1 位，即等于零，则返回零。

方法声明

```
public static long lowestOneBit(long i)
```

入口参数：i 为 long 值。

返回值：返回具有单个 1 位的 long 值，在指定值中最低位的 1 位的位置；否则，如果指定值本身等于零，则返回零。

示例

本示例通过 lowestOneBit()方法输出了 10 最低位的 1 位（最左边）的位置，具有单个 1 位的 long 值。

```
Long l = new Long(10);
long j = Long.lowestOneBit(l);
System.out.println(j);
```

2.2.16 parseLong 方法——将字符串解析为 long 值

该方法将字符串参数作为有符号的十进制整数进行解析。需要注意的是字符串中的字符都必须是十进制数字。

方法声明 1

```
public static int parseLong(String s) throws NumberFormatException
```

入口参数：s 为包含要解析的 Long 表示形式的 String。

返回值：十进制的 Long 值。

抛出异常

NumberFormatException：如果 String 不包含可解析的 Long。

注意

不允许将字符 L('\u004C')和 l('\u006C')作为类型指示符出现在字符串的结尾处，这一点在 Java 编程语言源代码中是允许的。

示例

本示例创建 Main 类，定义主方法 main，在主方法中定义 String 数组，实现将 Strring 类型数组中的元素转换为 Long 类型，并将各元素相加。

```
public class Main{                                          //创建类
public static void main(String[] args) {                   //定义主方法
    String str[] = {"15","20","45","35"};                  //定义 String 数组
    long sum = 0;                                          //定义 long 型变量 sum
    for(int i=0; i<str.length;i++){                        //循环遍历数组
        long mylong =Long.parseLong(str[i]);               //数组中元素转换为 long 型
        sum =sum+=mylong;                                  //将数组中的各元素相加
    }
    System.out.println("数组中各元素的和是："+sum);        //输出结果
}
}
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

将 string 参数解析为有符号的 long，基数由第二个参数指定。字符串中的字符必须为指定基数中的数字（由 Character.digit(char,int)是否返回一个非负值来确定），除非第一个字符为 ASCII 字符的减号'-'('\u002D')，它表示一个负值。

```
public static int parseLong(String s, int radix)
        throws NumberFormatException
```

入口参数

s：包含要解析的 long 表示形式的 String。

radix：解析 s 时使用的基数。

返回值：由指定基数中的字符串参数表示的 long。

抛出异常

NumberFormatException：如果 String 不包含可解析的 int。

注意

不允许将字符 L('\u004C')和 l('\u006C')作为类型指示符出现在字符串的结尾处，而这一点在 Java 编程语言源代码中是允许的，除非 L 或 l 以大于 22 的基数形式出现。

如果出现以下情形之一，则抛出 NumberFormatException 类型的异常。

- 第一个参数是 null 或零长度的字符串。
- radix 小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX。
- 任何字符串的字符都不是指定基数的数字，除非第一个字符是减号'-'('\u002d')，假定字符串的长度大于 1。
- 字符串表示的值不是 long 类型的值。

示例

本示例创建 3 个 String 对象，并通过 Long 类的 parseLong()方法将 String 类型转换为 Long 对象，并将转化后的结果输出。

```
String str1 = new String("0");
String str2 = new String("123");
String str3 = new String("-0");
long strResult1 = Long.parseLong(str1,10);
long strResult2 = Long.parseLong(str2,10);
long strResult3 = Long.parseLong(str3,10);
```

示例

本示例创建 2 个 String 对象，并通过 Long 类的 parseLong()方法将 String 类型转换为 Long 对象，并将转化后的结果输出。

```
String str1 = new String("2147483647");
String str2 = new String("-2147483648");
long strResult1 = Long.parseLong(str1,10);
long strResult2 = Long.parseLong(str2,10);
```

示例

本示例创建 2 个 String 对象，并通过 Long 类的 parseLong()方法将 String 类型转换为 Long 对象。由于字符串中不包含可解析的 Long，所以抛出异常。

```
String str4 = new String("99");
String str5 = new String("kona");
long strResult4 = Long.parseLong(str4,8);
long strResult5 = Long.parseLong(str5,10);
```

示例

本示例创建 3 个 String 对象，并通过 Long 类的 parseLong()方法将 String 类型转换为 Long 对象，并将返回值赋给 long 型变量。

```
String str1 = new String("-FF");
String str2 = new String("1100110");
String str3 = new String("kona");
long strResult1 = Long.parseLong(str1,16);
long strResult2 = Long.parseLong(str2,2);
long strResult3 = Long.parseLong(str3,27);
```

2.2.17 reverse 方法——反转长整数二进制补码的位顺序

返回通过反转指定 long 值的二进制补码表示形式中位的顺序而获得的值。

方法声明

```
public static long reverse(long i)
```

入口参数：i 为要反转指定的 Long 值。

返回值：返回通过反转指定 long 值中位的顺序而获得的值。

示例

本示例通过 reverse 方法，反转了 10 的二进制补码表示形式中位的顺序。

```
long i;
i = Long.reverse(10);
```

2.2.18 reverseBytes 方法——反转长整数字节的顺序

返回通过反转指定 long 值的二进制补码表示形式中字节的顺序而获得的值。

方法声明

```
public static long reverseBytes(long i)
```

入口参数：i 为要反转指定的 long 值。

返回值：返回通过反转指定 long 值中的字节而获得的值。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

示例

本示例通过 reverseBytes 方法，反转了 10 的二进制补码表示形式中字节的顺序。

```
long i;
i = Long.reverseBytes(10);
```

2.2.19 shortValue 方法——获取 short 值

以 short 类型返回该 Long 的值。

方法声明

```
public short shortValue();
```

返回值：转换为 short 类型后该对象表示的数值。

示例

本示例创建 Long 对象，并通过 shortValue()方法将 Long 对象转换为 long 变量，并将转换后的结果输出。

```
Long shorts = new Long(12L);
short str = shorts.shortValue();
```

2.2.20 signum 方法——获取长整数符号

返回指定 long 值的符号函数（如果指定值为负，则返回值-1；如果指定值为零，则返回 0；如果指定值为正，则返回 1）。

方法声明

```
public static int signum(long i)
```

入口参数：i 为要返回符号函数的整数。

返回值：返回指定 long 值的符号函数。

示例

本示例定义了 3 个整数，分别为正数、负数和 0，通过 signum()方法输出了它们的符号函数为：1、-1、0。

```
long i=10,j=-10,k=0;
System.out.println(Long.signum(i));
System.out.println(Long.signum(j));
System.out.println(Long.signum(k));
```

2.2.21 toBinaryString 方法——生成长整数的二进制字符串

以二进制（基数 2）无符号整数形式返回 long 参数的字符串表示形式。

方法声明

```
public static String toBinaryString(long i);
```

入口参数：i 为要转换成字符串的 long。

返回值：二进制（基数 2）参数表示的无符号 long 值的字符串表示形式。

注意

如果参数为负，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的二进制（基数 2）ASCII 数字字符串。如果无符号大小为零，则用单个零字符'0'表示它('\u0030')；否则，无符号大小表示形式中的第一个字符将不是零字符。字符'0'('\u0030')和'1'('\u0031')被用作二进制位。

示例

本示例创建 Long 对象，并通过 toBinaryString()方法获取到 10L 的二进制无符号 long 形式的字符串。

```
Long stri =new Long(10L);
String str=Long.toBinaryString(stri);
```

2.2.22 toHexString 方法——生成长整数的十六进制字符串

以十六进制（基数 16）无符号整数形式返回一个 long 参数的字符串表示形式。

方法声明

```
public static String toHexString(long i);
```

入口参数：i 为要转换成字符串的整数。

返回值：十六进制（基数 16）参数的无符号 long 值的字符串表示形式。

注意

如果参数为负，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的十六进制（基数 16）ASCII 数字字符串。如果无符号大小为零，则该数字由单个零字符'0'表示('\u0030')；否则，无符号大小表示形式中的第一个字符将不是零字符。

示例

本示例创建 Long 对象，并通过 toHexString()方法获取到整数 456 的十六进制无符号 long 形式的字符串。

```
Long stri =new Long(456);
String str=Long.toHexString(stri);
```

2.2.23 toOctalString 方法——生成长整数的八进制字符串

以八进制（基数 8）无符号整数形式返回一个 long 参数的字符串表示形式。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public static String toOctalString(long i);
```

入口参数：i 为要转换成字符串的整数。

返回值：八进制参数（基数 8）表示的无符号 long 值的字符串表示形式。

注意

如果参数为负，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的八进制（基数 8）ASCII 数字字符串。

如果无符号大小为零，则该数字用单个零字符'0'('\u0030')表示，否则无符号大小表示形式中的第一个字符将不是零字符。以下字符都用作八进制数字：01234567，这些是从'\u0030'～'\u0037'的字符。

示例

本示例创建 Long 对象，并通过 toOctalString()方法获取到整数 456 的八进制无符号 long 形式的字符串。

```
Long stri =new Long(456);
String str=Long.toOctalString(stri);
```

2.2.24 toString 方法——生成长整数的十进制字符串

返回表示指定 long 的 String 对象。该参数被转换为有符号的十进制表示形式，并作为字符串返回，该字符串与用该参数和基数 10 作为参数的 toString(long,int)方法所得到的值非常相似。

方法声明 1

```
public static String toString(long i);
```

入口参数：i 为要转换成字符串的 long。

返回值：十进制参数的字符串表示形式。

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的十进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str=Long.toString(stri);
```

方法声明 2

返回用第二个参数指定基数表示的第一个参数的字符串表示形式。

```
public static String toString(long i, int radix);
```

入口参数

i：要转换成字符串的 long。

radix：在字符串表示形式中使用的基数。

返回值：指定基数中参数的字符串表示形式。

注意

如果该基数小于 Character.MIN_RADIX，或大于 Character.MAX_RADIX，则使用基数 10。

如果第一个参数是负数，则结果的第一个元素是 ASCII 字符的减号'-'('\u002d')。如果第一个参数非负，则结果中不会出现符号字符。

结果的其余字符表示第一个参数的大小。如果大小为零，则用单个零字符'0'表示它('\u0030')；否则大小表示形式中的第一个字符将不是零字符。以下 ASCII 字符均被用作数字：0123456789abcdefghijklmnopqrstuvwxyz。这些是从'\u0030'到'\u0039'和从'\u0061'到'\u007a'的字符。

如果 radix 是 N，则这些字符的第一个 N 用作显示顺序中基数 N 的数字。因此，该数字的十六进制（基数 16）表示形式为 0123456789abcdef。如果需要使用大写字母，则可以在结果上调用 String.toUpperCase()方法：Long.toString(n,16).toUpperCase()。

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的二进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str=Long.toString(stri,2);
```

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的八进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str=Long.toString(stri,8);
```

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的十进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str=Long.toString(stri,10);
```

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的十六进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str=Long.toString(stri,16);
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 3

```
public String toString()
```

返回值：十进制参数的字符串表示形式。

示例

本示例创建 Long 对象，并通过 toString()方法获取整数 10 的十进制无符号 long 形式的字符串。

```
Long stri =new Long(10);
String str= stri.toString();
```

2.2.25 valueOf 方法——创建 Long 对象

返回一个表示指定的 long 值的 Long 实例。如果不需要新的 Long 实例，则通常应优先使用该方法，而不是构造方法 Long(long)，因为该方法有可能通过缓存经常请求的值而显著提高空间和时间性能。

方法声明 1

```
public static Long valueOf(long i);
```

入口参数：i 为一个 long 值。

返回值：表示 i 的 Long 实例。

示例

本示例创建 Long 对象，并通过 valueOf()方法获取 long 值 10L 的 Long 实例。

```
Long stri =new Long(10L);
long str=Long.valueOf(stri);
```

方法声明 2

返回保持指定 String 的值的 Long 对象。该参数被解释为表示一个有符号的十进制 long，该值与用该参数作为参数的 parseLong(java.lang.String)方法得到的值非常相似。结果是表示由字符串指定的整数值的 Long 对象。换句话说，此方法返回一个 Long 对象，它的值等于 new Long(Long.parseLong(s))。

```
public static Long valueOf(String s)
        throws NumberFormatException;
```

入口参数：s 为要解析的字符串。

返回值：包含由字符串参数表示的值的 Long 对象。

抛出异常

NumberFormatException：如果字符串不能解析为一个 long。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 10 的 Long 对象。

```
String stri =new String("10");
long str=Long.valueOf(stri);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值“明日科技”的 Long 对象，由于该值不能解析为 long，所以抛出异常。

```
String stri =new String("明日科技");
long str=Long.valueOf(stri);
System.out.println(str);
```

方法声明 3

返回从指定的 String 中提取的值的 Long 对象，并通过第二个参数指定基数。

```
public static Long valueOf(String s, int radix)
        throws NumberFormatException;
```

入口参数

s：要解析的字符串。

radix：解释 s 时使用的基数。

返回值：保持由指定基数中的字符串参数表示的值的 Long 对象。

抛出异常

NumberFormatException：如果字符串不能解析为一个整数。

注意

该方法将第一个参数解释为有符号的 long，基数由第二个参数指定，该值与用该参数作为参数的 parseLong(java.lang.String,int)方法得到的值非常类似。结果是表示字符串指定的 long 值的 Long 对象。换句话说，此方法返回一个 Long 对象，它的值等于：new Long(Long.parseLong(s,radix))。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 1010，以 2 为基数所表示的数值。

```
String stri =new String("1010");
long str=Long.valueOf(stri,2);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 0125，以 8 为基数所表示的数值。

```
String stri =new String("0125");
long str=Long.valueOf(stri,8);
```


PDF电子书基地 http://dayo1982.400gb.com

2.3 Short 类——短整型类

java.lang 包中的 short 类在对象中包装基本类型 short 的值。一个 Short 类型的对象只包含一个 short 类型的字段。此外，该类中提供了很多将 short 转换为 String，将 String 转换为 short 的方法。

2.3.1 构造方法——Short 类的构造方法

Short 类的构造方法和使用说明如下。

构 造 方 法	使 用 说 明
Short(short value)	构造一个 Short 对象，表示指定的 short 值
Short(String str)	构造一个 Short 对象，表示由 String 参数指示的 short 值

2.3.2 常量——Short 类的常量

Short 类的常量含义和数值说明如下。

常　量	含　义	数　值
MAX_VALUE	保存 short 可取的最大值常量	$2^{15}-1$
MIN_VALUE	保存 short 可取的最小值常量	-2^{15}
SIZE	用来以二进制补码表示 short 值的比特位数	—
TYPE	表示基本类型 short 的 Class 实例	—

2.3.3 compareTo 方法——比较短整数

比较两个 Short 对象所表示的数值。

方法声明

```
public int compareTo(Short anotherShort);
```

入口参数：anotherShort 是要比较的 Short。

返回值：如果该 Short 等于 Short 参数，则返回 0 值；如果 Short 在数字上小于 Short 参数，则返回小于 0 的值；如果 Short 在数字上大于 Short 参数，则返回大于 0 的值（有符号的比较）。

示例

本示例创建 2 个 Short 对象，应用 compareTo()方法比较 Short 对象的大小，将比较结果赋给 int 型变量 str。

```
Short com = new Short("125");
Short com1 = new Short("125");
int str = com.compareTo(com1);
```

2.3.4 decode 方法——字符串解码为 short 类型

将 String 解码为 Short。

方法声明

```
public static Short decode(String nm)
        throws NumberFormatException;
```

入口参数：nm 指定要解码的 string。

返回值：nm 所表示的 Short 值的 Short 对象。

抛出异常

NumberFormatException：如果 String 不包含可解析 short。

示例

本示例创建 String 对象，并通过 decode()方法将 String 对象的参数进行解码，并将返回结果赋给 Short 型变量 str。

```
String stri =new String("100");
Short str=Short.decode(stri);
```

2.3.5 doubleValue 方法——返回 double 数值

以 double 类型返回该 Short 的值。

方法声明

```
public double doubleValue();
```

返回值：转换为 double 类型后该对象表示的数值。

示例

本示例创建 Short 对象，并通过 doubleValue()方法将 Short 对象转换为 double 变量，并将转换后的结果赋给 double 变量 str。

```
short shor=125;
Short doubles = new Short(shor);
double str = doubles.doubleValue();
```

2.3.6 equals 方法——判断短整数对象相等

将此对象与指定对象比较。当且仅当参数不是 null，而是一个与该对象一样包含相同 short 值的 Short 对象时，结果才为 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 2 个 Short 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。

```
Short i = new Short("123");
Short j = new Short("123");
boolean b = i.equals(j);
System.out.println(b);
```

示例

本示例创建 1 个 Short 对象、1 个 String 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。Short 对象与此对象包含相同的 Short 值时，结果才为 true。

```
Short l = new Short ("123");
String s = "123";
boolean b = l.equals(s);
System.out.println(b);
b = l.equals(Short.valueOf(s));
System.out.println(b);
```

2.3.7 floatValue 方法——获取 float 数值

以 float 类型返回该 Short 的值。

方法声明

```
public float floatValue();
```

返回值：转换为 float 类型后该对象表示的数值。

示例

本示例创建 Short 对象，并通过 floatValue()方法将 Short 对象转换为 float 变量。并将转换后的结果赋给 float 型变量 str。

```
short shor=125;
Short floats = new Short(shor);
float str = floats.floatValue();
```

2.3.8 hashCode 方法——生成短整数的哈希码

此方法返回此 Short 的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回此 Short 的哈希码。

示例

本示例创建 1 个 Short 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Short i = new Short ("10");
int j = i.hashCode();
System.out.println(j);
```

2.3.9 intValue()方法——获取 int 值

该方法以 int 类型返回该 Short 的值。

方法声明

```
public int intValue();
```

返回值：转换为 int 类型后该对象表示的数值。

示例

本示例创建 Short 对象，并通过 intValue()方法将 Short 对象转换为 int 变量。并将转换后的结果输出。

```
short shor=125;
Short ints = new Short(shor);
int str = ints.intValue();
```

2.3.10 longValue 方法——获取 long 值

以 Long 类型返回该 Short 的值。

方法声明

```
public long longValue();
```

返回值：转换为 long 类型后该对象表示的数值。

示例

本示例创建 Short 对象，并通过 longValue()方法将 Short 对象转换为 long 变量。并将转换后的结果输出。

```
Short Shorts = new Short("12345");        //以 String 类型变量作为参数创建对象
long str = Shorts.longValue();            //返回值为 long 类型
```

2.3.11 parseShort 方法——将字符串解析为 short 值

该方法将字符串参数作为有符号的十进制 long 进行解析。需要注意的是字符串中的字符都必须是十进制数字。

方法声明 1

```
public static short parseShort(String s)
    throws NumberFormatException
```

入口参数：s 包含要解析的 Short 表示形式的 String。

返回值：十进制的 Short 值。

抛出异常

NumberFormatException：如果 String 不包含可解析的 Short。

PDF电子书基地 http://dayo1982.400gb.com

注意

该字符串中的字符必须都是十进制数字，除非第一个字符是表示负值的 ASCII 符号中的负号'-'('\u002D')。返回得到的 short 值，此值与用该参数和基数 10 作为参数的 parseShort(java.lang.String,int)方法得到的值相同。

示例

本示例创建 Main 类，定义主方法 main，在主方法中定义 String 数组，实现将 Strring 类型数组中的元素转换为 Short 类型，并将各元素相加。

```
public class Main{                                          //创建类
public static void main(String[] args) {                    //定义主方法
    String str[] = {"15","20","45","35"};                   //定义 String 数组
    short sum = 0;                                          //定义 Short 型变量 sum
    for(int i=0; i<str.length;i++){                         //循环遍历数组
        Short myShort =Short.parseShort(str[i]);            //数组中元素转换为 Short 型
        sum =sum+=myShort;                                  //将数组中的各元素相加
    }
    System.out.println("数组中各元素的和是："+sum);        //输出结果
}
}
```

方法声明 2

将 string 参数解析为有符号的 Short，基数由第二个参数指定。

```
public static short parseShort(String s, int radix)
    throws NumberFormatException
```

入口参数

s：包含要解析的 Short 表示形式的 String。

radix：解析 s 时使用的基数。

返回值：由指定基数中的字符串参数表示的 short。

抛出异常

NumberFormatException：如果 String 不包含可解析的 int。

注意

该字符串中的字符必须都是指定基数（这取决于 Character.digit(char,int)是否返回非负值）的数字，除非第一个字符是表示负值的 ASCII 符号中的负号“-”('\u002D')。返回得到的 byte 值。

如果出现以下情形之一，则抛出 NumberFormatException 类型的异常：

第一个参数是 null 或零长度的字符串。

基数小于 Character.MIN_RADIX 或大于 Character.MAX_RADIX。

除了在字符串长度超过 1 的情况下第一个字符可能是负号“-”('\u002D')之

外，字符串的任何字符都不是指定基数的数字。

字符串所表示的值不是 short 类型的值。

示例

本示例通过 Short 类的 parseShort()方法将 String 类型转换为 Short 对象。并将转化后的结果输出。

```
public class Main{                                          //创建类
public static void main(String[] args) {                   //定义主方法
Short strResult1 = Short.parseShort("1010",2);             //定义字符串 1010，基数为 2
System.out.println("strResult1： "+strResult1);
Short strResult2 = Short.parseShort("0123",8);             //定义字符串 0123，基数为 8
System.out.println("strResult2： "+strResult2);
Short strResult3 = Short.parseShort("10",10);              //定义字符串 10，基数为 10
System.out.println("strResult3： "+strResult3);
Short strResult4 = Short.parseShort("a",16);               //定义字符串 a，基数为 16
System.out.println("strResult4： "+strResult4);
}
}
```

2.3.12 reverseBytes 方法——反转短整数字节的顺序

返回通过反转指定 short 值的二进制补码表示形式中字节的顺序而获得的值。

方法声明

```
public static short reverseBytes(short i)
```

入口参数：i 要反转指定的 short 值。

返回值：返回通过反转（或者交换，效果相同）指定 short 值中的字节而获得的值。

示例

本示例通过 reverseBytes 方法，反转了 10 的二进制补码表示形式中字节的顺序。

```
short i;
short j = 10;
i = Short.reverseBytes(j);
```

2.3.13 shortValue 方法——获取 short 值

以 short 形式返回此 Short 的值。

方法声明

```
public short shortValue()
```

返回值：以 short 形式返回此 Short 的值。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例定义了 Short 和 short 变量，通过 shortValue 方法，以 short 形式返回此 Short 的值。

```
Short i = 10;
short j = i.shortValue();
```

2.3.14 toString 方法——生成短整数的十进制字符串

返回表示指定 Short 的 String 对象。该参数被转换为有符号的十进制表示形式，并作为字符串返回，该字符串与用该参数和基数 10 作为参数的 toString(Short,int)方法所得到的值非常相似。

方法声明 1

```
public static String toString(Short s);
```

入口参数：s 为要转换成字符串的 Short。

返回值：参数（十进制）表示的 short 值。

示例

本示例创建 Short 对象，并通过 toString()方法获取到整数 10 的十进制无符号 Short 形式的字符串。

```
Short stri =new Short(10);
String str=Short.toString(stri);
```

方法声明 2

```
public String toString()
```

返回值：该对象值（十进制）的字符串表示形式。

示例

本示例创建 Short 对象，并通过 toString()方法获取到整数 10 的十进制无符号 Short 形式的字符串。

```
Short stri =new Short ("10");
String str= stri.toString();
```

2.3.15 valueOf 方法——创建 Short 对象

返回一个表示指定的 Short 值的 Short 实例。如果不需要新的 Short 实例，则通常应优先使用该方法，而不是构造方法 Short(Short)，因为该方法有可能通过缓存经常请求的值而显著提高空间和时间性能。

方法声明 1

```
public static Short valueOf(Short s);
```

入口参数：s 为一个 Short 值。

返回值：表示 s 的 Short 实例。

示例

本示例创建 Short 对象，并通过 valueOf()方法获取 Short 值 10L 的 Short 实例。

```
Short stri =new Short("10");
Short str=Short.valueOf(stri);
```

方法声明 2

返回一个保持指定 String 所给出的值的 Short 对象。该参数被解释为表示一个有符号的十进制 short，此值与用该参数作为参数的 parseLong(java.lang.String)方法得到的值相同。结果是一个表示该字符串所指定的 short 值的 Short 对象。

```
public static Short valueOf(String s)
        throws NumberFormatException;
```

入口参数：s 为要解析的字符串。

返回值：保持字符串参数所表示的值的 Short 对象。

抛出异常

NumberFormatException：如果字符串不能解析为一个 Short。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 10 的 Short 对象。

```
String stri =new String("10");
Short str=Short.valueOf(stri);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值“明日科技”的 Short 对象，由于该值不能解析为 Short，所以抛出异常。

```
String stri =new String("明日科技");
Short str=Short.valueOf(stri);
System.out.println(str);
```

方法声明 3

返回从指定的 String 中提取的值的 Short 对象，并通过第二个参数指定基数。

```
public static Short valueOf(String s, int radix)
        throws NumberFormatException
```

入口参数

s：要解析的字符串。

PDF电子书基地 http://dayo1982.400gb.com

radix：解释 s 时使用的基数。

返回值：保持指定基数中的字符串参数所表示的值的 Short 对象。

抛出异常：

NumberFormatException：如果字符串不能解析为一个 short。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 0110，以 8 为基数所表示的数值。

```
String stri =new String("0110");
short str=Short.valueOf(stri,8);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 1010，以 2 为基数所表示的数值。

```
String stri =new String("1010");
short str=Short.valueOf(stri,2);
```

2.4 Boolean 类——布尔类

java.lang 包中的 boolean 类将基本类型为 boolean 的值包装在一个对象中。一个 Boolean 类型的对象只包含一个类型为 boolean 的字段。该类还为 boolean 和 String 的相互转换提供了方法。

2.4.1 构造方法——Boolean 类的构造方法

Boolean 类的构造方法

构 造 方 法	使 用 说 明
Boolean(boolean value)	构造一个表示 value 参数的 Boolean 对象
Boolean(String str)	当参数不为 NULL 且在忽略大小写时等于“true”，则分配一个表示 true 值的 Boolean 对象。否则分配一个表示 false 值的 Boolean 对象

2.4.2 常量——Boolean 类的常量

Boolean 类的常量

常　　量	含　　义
FALSE	表示对应基值 false 的 Boolean 对象
TRUE	表示对应基值 true 的 Boolean 对象
TYPE	表示基本类型 boolean 的 Class 对象

2.4.3 booleanValue 方法——获取 boolean 值

将此 Boolean 对象的值作为基本布尔值返回。

方法声明

```
public boolean booleanValue();
```

返回值：转换为 boolean 类型后该对象表示的数值。

示例

本示例将 Boolean 对象的值返回基本布尔值。

```
Boolean bool = new Boolean("true");
boolean b = bool.booleanValue();
```

示例

本示例在项目中创建类 Main，在主方法中以不同的构造函数创建 boolean 对象，并调用 boolean()方法将创建的对象重新转换为 boolean 数据输出。

```
public class Main{                                          //创建类
public static void main(String[] args) {                   //定义主方法
      Boolean b1 = new Boolean(true);                       //创建 boolean 对象
      Boolean b2 = new Boolean("ok");                       //创建 boolean 对象
      System.out.println("b1:"+b1.booleanValue());
      System.out.println("b2:"+b2.booleanValue());
}
}
```

2.4.4 compareTo 方法——比较布尔值

将此 Boolean 实例与其他实例进行比较。

方法声明

```
public int compareTo(Boolean b)
```

入口参数：b 为要进行比较的 Boolean 实例。

返回值：如果对象与参数表示的布尔值相同，则返回零；如果此对象表示 true，参数表示 false，则返回一个正值；如果此对象表示 false，参数表示 true，则返回一个负值。

抛出异常

NullPointerExceptio：如果参数为 null。

示例

本示例创建 3 个 Boolean 对象，应用 compareTo()方法进行比较，将比较结果赋给 int 型变量 i 并输出结果。

```
Boolean b1 = true;          //创建 Boolean 对象
Boolean b2 = false;         //创建 Boolean 对象
Boolean b3 = true;          //创建 Boolean 对象
```


PDF电子书基地 http://dayo1982.400gb.com

```
int i;
i = b1.compareTo(b2);      //b1 和 b2 进行比较
System.out.println(i);
i = b2.compareTo(b1);      //b2 和 b1 进行比较
System.out.println(i);
i = b1.compareTo(b3);      //b1 和 b3 进行比较
System.out.println(i);
```

2.4.5 equals 方法——判断相等

当且仅当参数不是 null，而是一个与此对象一样，都表示同一个 Boolean 值的 boolean 对象时，才返回 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：当且仅当参数不是 null，而是一个与此对象一样，都表示同一个 Boolean 值的 boolean 对象时，才返回 true。

示例

本示例创建 2 个 Short 对象，通过 equals 进行比较,将比较的结果赋给 b 并输出。

```
Boolean b1 = true;         //创建 Boolean 对象
Boolean b2 = false;        //创建 Boolean 对象
Boolean b3 = true;         //创建 Boolean 对象
boolean b;
b = b1.equals(b2);         //b1 是否和 b2 相等
System.out.println(b);
b = b1.equals(b3);         //b1 是否和 b3 相等
System.out.println(b);
```

2.4.6 getBoolean 方法——获取布尔类型的系统属性值

当且仅当以参数命名的系统属性存在，且等于"true"字符串时，才返回 true。（从 JavaTM 平台的 1.0.2 版本开始，字符串的测试不再区分大小写。）通过 getProperty 方法可访问系统属性，此方法由 System 类定义。如果没有以指定名称命名的属性或者指定名称为空或 null，则返回 false。

方法声明

```
public static boolean getBoolean(String name)
```

入口参数：obj 为要比较的对象。

返回值：系统属性的 boolean 值。

示例

本示例系统属性"sun.arch.data.model"的 boolean 值。

```
String nm = "sun.arch.data.model";
boolean b = Boolean.getBoolean(nm);
```

2.4.7 hashCode 方法——生成布尔对象的哈希码

返回该 Boolean 对象的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回该 Boolean 对象的哈希码。

示例

本示例创建 1 个 Short 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Boolean b = true;
int i = b.hashCode();
System.out.println(j);
```

2.4.8 parseBoolean 方法——将字符串解析成 boolean 值

将字符串参数解析为 boolean 值。如果 String 参数不是 null 且在忽略大小写时等于"true"，则返回的 boolean 表示 true 值。

方法声明

```
public static boolean parseBoolean(String s)
```

入口参数：s 为包含要解析的布尔表达式的 String。

返回值：String 参数所表示的布尔值。

示例

本示例通过 parseBoolean()方法返回 String 参数所表示的布尔值。

```
boolean b ;
b = Boolean.parseBoolean("True");
b = Boolean.parseBoolean("False")
```

示例

本示例通过 parseBoolean()方法返回 String 参数所表示的布尔值。当 String 为“yes”时返回 false。

```
boolean b ;
b = Boolean.parseBoolean("Yes");
```

2.4.9 toString 方法——生成布尔值的字符串

返回表示该布尔值的 String 对象。如果该对象表示 true 值，则返回等于"true"的字符串。否则返回等于"false"的字符串。

PDF电子书基地 http://dayo1982.400gb.com

方法声明 1

```
public String toString()
```

返回值：此对象的字符串表示形式。

示例

本示例创建 Boolean 对象，并通过 toString()方法返回字符串表示形式。

```
Boolean b = false;
String s = b.toString();
System.out.println(s);
```

方法声明 2

返回参数的字符串表示形式。

```
public static String toString(boolean b)
```

入口参数：b 为要转换的布尔值。

返回值：指定 boolean 值的字符串表达式。

示例

本示例通过 toString()方法返回 boolean 型 b 的字符串表示形式。

```
boolean b = true;
String s = Boolean.toString(b);
```

2.4.10 valueOf 方法——创建布尔对象

返回一个表示指定 boolean 值的 Boolean 实例。如果指定的 boolean 值为 true，则此方法返回 Boolean.TRUE；如果为 false，则返回 Boolean.FALSE。如果不需要新的 Boolean 实例，则应优先使用此方法，而不是构造方法 Boolean(boolean)，因为此方法有可能大大提高空间和时间性能。

方法声明 1

```
public static Boolean valueOf(boolean b)
```

返回值：一个表示 b 的 Boolean 实例。

示例

本示例通过 valueOf()方法返回 boolean b 的 Boolean 实例。

```
boolean i = true;
Boolean b = Boolean.valueOf(i);
```

方法声明 2

返回一个指定参数值的 Boolean 实例。

```
public static Boolean valueOf(String s)
```

入口参数：s 为字符串。

返回值：该字符串所表示的 Boolean 值。

示例

本示例通过 valueOf()方法返回 String 类型的 Boolean 实例。

```
String s = "true";
Boolean b = Boolean.valueOf(s);
```

2.5 Byte 类——字节对象

java.lang 包中的 byte 类将基本类型 byte 包装在一个对象中。一个 Byte 类型的对象只包含一个类型为 byte 的字段。该类中提供了处理 byte 时非常有用的其他一些常用常量和方法。

2.5.1 构造方法——Byte 类的构造方法

Byte 类的构造方法

构 造 方 法	使 用 说 明
Byte(byte value)	构造一个表示指定的 byte 值的 Byte 对象
Byte(String str)	构造一个表示 String 参数所指示的 byte 值的 Byte 对象

2.5.2 常量——Byte 类的常量

Byte 类的常量

常 量	含 义	数 值
MAX_VALUE	保存 byte 类型可取的最大值	2^7-1
MIN_VALUE	保存 byte 类型可取的最小值	-2^7
SIZE	用于以二进制补码形式表示 byte 值的位数	
TYPE	表示基本类型 byte 的 Class 实例	

2.5.3 compareTo 方法——比较字节对象

在数字上比较两个 Byte 对象。

方法声明

```
public int compareTo(Byte anotherByte)
```

入口参数：anotherByte 要比较的 Byte。

返回值：如果此 Byte 等于参数 Byte，则返回 0；如果此 Byte 在数字上小于参数 Byte，则返回小于 0 的值；如果此 Byte 在数字上大于参数 Byte，则返回大于 0 的值（有符号比较）。

示例

本示例创建 3 个 Byte 对象，应用 compareTo()方法比较 Byte 对象的大小，

PDF电子书基地 http://dayo1982.400gb.com

将比较结果赋给 int 型变量 i 并输出。

```
Byte b1 = new Byte("10");          //创建 Byte 对象
Byte b2 = new Byte("20");          //创建 Byte 对象
Byte b3 = new Byte("10");          //创建 Byte 对象
int i;
i = b1.compareTo(b2);              //b1 与 b2 进行比较
System.out.println(i);
i = b2.compareTo(b1);              //b2 与 b1 进行比较
System.out.println(i);
i = b1.compareTo(b3);              //b1 与 b3 进行比较
System.out.println(i);
```

2.5.4 decode 方法——将字符串解码为 Byte 值

将 String 解码为 Byte。

方法声明

```
public static Byte decode(String nm)
        throws NumberFormatException
```

入口参数：nm 为要解码的 String。

返回值：保存由 nm 表示的 byte 值的 Byte 对象。

抛出异常

NumberFormatExceptio：如果 String 不包含一个可解析的 byte。

注意

接受通过以下语法给出的十进制、十六进制和八进制数：

DecodableString:

Signopt DecimalNumeral

Signopt 0x HexDigits

Signopt 0X HexDigits

Signopt # HexDigits

Signopt 0 OctalDigits

对（可选）负号和/或基数说明符（“0x”、“0X”、“#”或前导零）后面的字符序列进行解析就如同使用带指定基数（10、16 或 8）的 Byte.parseByte 方法一样。该字符序列必须表示一个正值，否则将抛出 NumberFormatException。如果指定 String 的第一个字符是负号，则结果将被求反。该 String 中不允许出现空白字符。

示例

本示例通过 decode()方法将 s 解码成 Byte 型值。

```
String s = "0x64";
Byte b = Byte.decode(s);
System.out.println(b);
```

示例

本示例通过 decode()方法将 s 解码成 Byte 型值。

```
String s = "123";
Byte b = Byte.decode(s);
```

2.5.5 doubleValue 方法——获取 double 值

以一个 double 返回此 Byte 的值。

方法声明

```
public double doubleValue();
```

返回值：转换为 double 类型后该对象表示的数值。

示例

本示例创建 Byte 对象，并通过 doubleValue()方法将 Byte 对象转换为 double 变量。并将转换后的结果赋给 double 变量 d。

```
Byte b=100;
double d = b.doubleValue();
System.out.println(d);
```

2.5.6 equals 方法——判断字节相等

将此对象与指定对象比较。当且仅当参数不为 null，而是一个与此对象一样包含相同 Byte 值的 byte 对象时，结果才为 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：当且仅当参数不为 null，而是一个与此对象一样包含相同 Byte 值的 byte 对象时，结果才为 true。

示例

本示例创建 3 个 Byte 对象，通过 equals 进行比较,将比较的结果赋给 b 并输出。

```
Byte b1 = 10;                    //创建 Byte 对象
Byte b2 = 20;                    //创建 Byte 对象
Byte b3 = 10;                    //创建 Byte 对象
boolean b;
b = b1.equals(b2);               //b1 是否和 b2 相等
System.out.println(b);
b = b1.equals(b3);               //b1 是否和 b3 相等
System.out.println(b);
```


PDF电子书基地 http://dayo1982.400gb.com

2.5.7 floatValue 方法——获取 float 值

以 float 类型返回该 Byte 的值。

方法声明

```
public float floatValue();
```

返回值：转换为 float 类型后该对象表示的数值。

示例

本示例创建 Byte 对象，并通过 floatValue()方法将 Byte 对象转换为 float 变量。并将转换后的结果赋给 float 型变量 str。

```
Byte b1 = 10;  //创建 Byte 对象
float f = b1.floatValue();
System.out.println(f);
```

2.5.8 hashCode 方法——生成字节对象的哈希码

返回该 Byte 对象的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回该 Byte 对象的哈希码。

示例

本示例创建 1 个 Byte 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Byte b1 = 10;  //创建 Byte 对象
int i = b1.hashCode();
System.out.println(i);
```

2.5.9 intValue 方法——获取 int 值

该方法作为一个 int 返回此 Byte 的值。

方法声明

```
public int intValue();
```

返回值：转换为 int 类型后该对象表示的数值。

示例

本示例通过 Byte 类的 byteValue()方法将 byte 类型变量转换成 int 型变量。

```
byte by = 123;
Byte b = new Byte(by);
int it = b.byteValue();
```

2.5.10 longValue 方法——获取 long 值

以 Long 类型返回该 Byte 的值。

方法声明

```
public long longValue();
```

返回值：转换为 long 类型后该对象表示的数值。

示例

本示例创建 Byte 对象，并通过 longValue()方法将 Byte 对象转换为 long 变量，并将转换后的结果输出。

```
Byte b1 = 10;  //创建 Byte 对象
long i = b1.longValue();
System.out.println(i);
```

2.5.11 parseByte 方法——将字符串解析为 byte 值

将 string 参数解析为有符号的十进制 byte。除了第一个字符可以是表示负值的 ASCII 负号'-'('\u002D')之外，该字符串中的字符必须都是十进制数字。

方法声明 1

```
public static byte parseByte(String s)
        throws NumberFormatException;
```

入口参数：s 为要解析的包含 byte 表示形式的 String。

返回值：以十进制的参数表示的 byte 值。

抛出异常

NumberFormatException：如果该 string 不包含一个可解析的 byte。

示例

本示例通过 Byte 类的 byteValue()方法将 String 对象的参数 12 转换成十进制的 byte 型变量。

```
String str = new String("12");
byte it = Byte.parseByte(str);
```

方法声明 2

将 string 参数解析为一个有符号的 byte，其基数由第二个参数指定。

```
public static byte parseByte(String s, int radix)
        throws NumberFormatException;
```

入口参数

s：要解析的包含 byte 表示形式的 String。

radix：解析 s 时使用的基数。

返回值：以指定基数表示的 string 参数所表示的 byte 值。

抛出异常

NumberFormatException：如果该 string 不包含一个可解析的 byte。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例通过 Byte 类的 byteValue()方法将 String 对象的参数 1010 转换成十进制的 byte 型变量，设置其基数为 2。

```
String str = new String("1010");
byte it = Byte.parseByte(str,2);
```

示例

本示例通过 Byte 类的 byteValue()方法将 String 对象的参数 0125 转换成十进制的 byte 型变量，设置其基数为 8。

```
String str = new String("0125");
byte it = Byte.parseByte(str,8);
```

2.5.12 shortValue 方法——获取 short 值

以 short 类型返回该 Byte 的值。

方法声明

```
public short shortValue();
```

返回值：转换为 short 类型后该对象表示的数值。

示例

本示例创建 Byte 对象，并通过 shortValue()方法将 Byte 对象转换为 short 变量。并将转换后的结果输出。

```
Byte b1 = 10;  //创建 Byte 对象
short i = b1.shortValue();
System.out.println(i);
```

2.5.13 toString 方法——生成字节值的十进制字符串

返回表示指定 byte 的一个新 String 对象。假定基数为 10。

方法声明 1

```
public static String toString(byte b);
```

入口参数：b 为要转换的 byte。

返回值：指定 byte 的字符串表示形式。

示例

本示例通过 Byte 类的 toString()方法将 Byte 对象的参数 125 转换成十进制的 String 型变量。

```
Byte str = new Byte("125");
String it = Byte.toString(str);
```

方法声明 2

```
public String toString()
```

返回值：以基数 10 表示的此对象值的字符串表示形式。

示例

本示例创建 Byte 对象，并通过 toString()方法返回 Byte 的字符串表示形式。

```
Byte b1 = 10;  //创建 Byte 对象
String s = b1.toString();
System.out.println(s);
```

2.5.14 valueOf 方法——创建 Byte 对象

返回表示指定 byte 值的一个 Byte 实例。如果不需要新的 Byte 实例，则通常应优先使用此方法，而不是构造方法 Byte(byte)，因为该方法有可能通过缓存经常请求的值来显著提高空间和时间性能。

方法声明 1

```
public static Byte valueOf(byte b)
```

入口参数：b 为一个 byte 值。

返回值：表示 b 的 Byte 实例。

示例

本示例创建 Byte 对象，并通过 valueOf()方法获取 byte 的 Byte 实例。

```
byte b1 = 10;
Byte b = Byte.valueOf(b1);  //创建 Byte 对象
System.out.println(b);
```

方法声明 2

返回一个保持指定 String 所给出的值的 Byte 对象。参数被解释为表示一个有符号的十进制的 byte，正如将该参数指定给 parseByte(java.lang.String)方法一样。结果是一个表示该 string 所指定的 byte 值的 Byte 对象。换句话说，该方法返回一个等于以下代码的值的 Byte 对象：new Byte(Byte.parseByte(s))。

```
public static Byte valueOf(String s)
    throws NumberFormatException
```

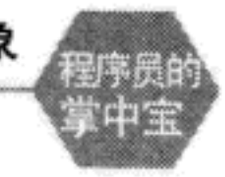
入口参数：s 为要解析的字符串。

返回值：保存 string 参数所表示的值的 Byte 对象。

抛出异常

NumberFormatException：如果该 String 不包含一个可解析的 byte。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 10 的 Byte 对象。

```
String s =new String("10");
Byte b = Byte.valueOf(s);
System.out.println(b);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值“明日科技”的 Byte 对象，由于该值不能解析为 Byte，所以抛出异常。

```
String s =new String("明日科技");
Byte b = Byte.valueOf(s);
System.out.println(b);
```

方法声明 3

返回从指定的 String 中提取的值的 Byte 对象，并通过第二个参数指定基数。

```
public static Long valueOf(String s, int radix)
        throws NumberFormatException;
```

入口参数

s：要解析的字符串。

radix：解释 s 时使用的基数。

返回值：保持由指定基数中的字符串参数表示的值的 Long 对象。

抛出异常

NumberFormatException：如果字符串不能解析为一个整数。

注意

该值是在用第二个参数所给定的基数对指定字符串进行解析时提取的。第一个参数被解释为用第二个参数所指定的基数表示一个有符号的 byte，正如将该参数指定给 parseByte(java.lang.String,int)方法一样。结果是一个表示该 string 所指定的 byte 值的 Byte 对象。换句话说，该方法返回一个等于以下代码的值的 Byte 对象： new Byte(Byte.parseByte(s,radix))。

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 1010，以 2 为基数所表示的数值。

```
String s = new String("1010");
Byte b = Byte.valueOf(s,2);
System.out.println(b);
```

示例

本示例创建 String 对象，并通过 valueOf()方法获取 String 值 0125，以 8 为基数所表示的数值。

```
String s = new String("0125");
Byte b = Byte.valueOf(s,8);
System.out.println(b);
```

2.6 Character 类——字符类

java.lang 包中的 character 类在对象中包装了一个基本类型 char 的值。Character 类型的对象包含类型为 char 的单个字段。

2.6.1 构造方法——Character 类的构造方法

语法：

```
Character(char value)
```

创建一个新分配的 Character 对象，用以表示指定的 char 值。

2.6.2 常量——Character 类的常量

Character 类的方法有很多，由于篇幅有限，下面列出 Character 类中部分常量，如下表所示。

Character 类的常量

常　量	含义数值
COMBINING_SPACING_MARK	Unicode 规范中的常规类型“Mc”
CONNECTOR_PUNCTUATION	Unicode 规范中的常规类型“Pc”
CONTROL	Unicode 规范中的常规类别“Cc”
CURRENCY_SYMBOL	Unicode 规范中的常规类别“Sc”
DASH_PUNCTUATION	Unicode 规范中的常规类别“Pd”
DECIMAL_DIGIT_NUMBER	Unicode 规范中的常规类别“Nd”
DIRECTIONALITY_RIGHT_TO_LEFT	Unicode 规范中的强双向字符类型“R”
DIRECTIONALITY_UNDEFINED	未定义的双向字符类型
DIRECTIONALITY_WHITESPACE	Unicode 规范中的非特定双向字符类型“WS”
ENCLOSING_MARK	Unicode 规范中的常规类别“Me”
END_PUNCTUATION	Unicode 规范中的常规类别“Pe”
FINAL_QUOTE_PUNCTUATION	Unicode 规范中的常规类别“Pf”
FORMAT	Unicode 规范中的常规类别“Cf”
INITIAL_QUOTE_PUNCTUATION	Unicode 规范中的常规类别“Pi”
LETTER_NUMBER	Unicode 规范中的常规类别“Nl”
LINE_SEPARATOR	Unicode 规范中的常规类别“Zl”

PDF电子书基地 http://dayo1982.400gb.com

续表

常　量	含 义 数 值
LOWERCASE_LETTER	Unicode 规范中的常规类别“Ll”
MAX_CODE_POINT	Unicode 代码点的最大值
MAX_HIGH_SURROGATE	UTF-16 编码中的 Unicode 高代理项代码单元的最大值
MAX_LOW_SURROGATE	UTF-16 编码中的 Unicode 低代理项代码单元的最大值
MAX_SURROGATE	UTF-16 编码中的 Unicode 代理项代码单元的最大值
MATH_SYMBOL	Unicode 规范中的常规类型“Sm”
MAX_RADIX	可用于与字符串相互转换的最大基数
TYPE	表示基本类型 char 的 Class 实例
UNASSIGNED	Unicode 规范中的常规类别“Cn”
UPPERCASE_LETTER	Unicode 规范中的常规类别“Lu”

2.6.3 charCount 方法——计算指定字符代码点的数量

确定表示指定字符（Unicode 代码点）所需的 char 值的数量。

方法声明

```
public static int charCount(int codePoint)
```

入口参数：codePoint 为要测试的字符（Unicode 代码点）。

返回值：如果字符是一个有效的增补字符，则返回 2；否则返回 1。

说明

如果指定字符等于或大于 0x10000，则该方法返回的值为 2。否则，该方法返回的值为 1。

该方法没有验证指定的字符是否为一个有效的 Unicode 代码点。如有必要，调用者必须使用 isValidCodePoint 验证字符值。

示例

本示例通过 Character 类的 charCount()方法，判断指定字符“0x10000”所需 char 值的数量。并将返回值赋值给 int 类型变量 cha。

```
int c1 = 0x10000;
int cha = Character.charCount(c1);
```

2.6.4 charValue 方法——获取 char 值

获取 Character 对象的值。

方法声明

```
public char charValue();
```

返回值：此对象表示的基本 char 值。

示例

本示例创建 Character 对象，通过 Character 类的 charValue()方法获取对象的 char 值。并将返回值赋值给 char 类型变量 cha。

```
Character c1 = new Character('a');
char cha = c1.charValue();
```

2.6.5 codePointAt 方法——获取字符数组元素的代码点

返回 char 数组的给定索引上的代码点。

方法声明 1

```
public static int codePointAt(char[] a,int index);
```

入口参数

a：char 数组。

index：要转换的 char 数组中的 char 值（Unicode 代码单元）的索引。

返回值：给定索引上的 Unicode 代码点。

抛出异常

NullPointerException：如果 a 为 null。

IndexOutOfBoundsException：如果 index 值为负或不小于 char 数组的长度。

说明

如果 char 数组中的给定索引上的 char 值在高代理项范围内，则下一个索引的长度小于 char 数组的长度，如果下一个索引上的 char 值在低代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回给定索引上的 char 值。

示例

本示例创建 char 数组，通过 Character 类的 codePoint()方法获取 char 数组中指定索引位置的 char 值。并将返回值赋值给 int 类型变量 cha。

```
char[] c1 = {'明','日','科','技'};
int cha = Character.codePointAt(c1,2);
```

方法声明 2

返回 char 数组的给定索引上的代码点，该数组中只有那些具有小于 limit 的 index 值的数组元素可以使用。

```
public static int codePointAt(char[] a, int index, int limit);
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：

a：char 数组。

index：要转换的 char 数组中的 char 值（Unicode 代码单元）的索引。

limit：可以在 char 数组中使用的最后一个数组元素后面的索引。

返回值：给定索引上的 Unicode 代码点。

抛出异常：

NullPointerException：如果 a 为 null。

IndexOutOfBoundsException：如果 index 参数为负或不小于 limit 参数，或者 limit 参数为负或大于 char 数组的长度。

说明

如果 char 数组中的给定索引上的 char 值在高代理项范围内，则下一个索引小于 limit，如果下一个索引上的 char 值在低代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回给定索引上的 char 值。

示例

本示例创建 char 数组，通过 Character 类的 codePoint()方法获取 char 数组中指定索引位置的 char 值。并设置参数 limit 的值，将返回值赋值给 int 类型变量。

```
char[] c1 = {'明','日','日','科','技'};
int cha0 = Character.codePointAt(c1,0,1);
int cha1 = Character.codePointAt(c1,1,2);
int cha2 = Character.codePointAt(c1,2,3);
int cha3 = Character.codePointAt(c1,3,4);
int cha4 = Character.codePointAt(c1,4,5);
```

2.6.6 codePointBefore 方法——获取字符数组索引前一个元素的代码点

返回 char 数组的给定索引前面的代码点。

方法声明 1

```
public static int codePointBefore(char[] a, int index)
```

入口参数

a：char 数组。

index：应该返回的代码点后面的索引。

返回值：返回 char 数组的给定索引前面的代码点。

抛出异常

NullPointerException：如果 a 为 null。

IndexOutOfBoundsException：如果 index 值为负或不小于 char 数组的长度。

说明

如果 char 数组中的(index−1)上的 char 值在低代理项范围内，则(index−2)为非负，如果 char 数组中的(index−2)上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回(index−1)上的 char 值。

示例

本示例创建 char 数组，通过 Character 类的 codePointBefore()方法获取 char 数组中指定索引位置前面的代码点。并将返回值赋值给 int 类型变量 cha。

```
char[] c1 = {'明','日','科','技'};
int cha = Character. codePointBefore(c1,2);
```

方法声明 2

返回 char 数组的给定索引前面的代码点，该数组中只有那些具有大于等于 start 的 index 值的数组元素可以使用。

```
public static int codePointBefore(char[] a, int index, int start)
```

入口参数

a：char 数组。

index：应该返回的代码点后面的索引。

start：char 数组中第一个数组元素的索引。

返回值：返回 char 数组的给定索引前面的代码点。

抛出异常

NullPointerException：如果 a 为 null。

IndexOutOfBoundsException：如果 index 参数不大于 start 参数或大于 char 数组的长度，或者 start 参数为负或小于 char 数组的长度。

说明

如果 char 数组中的(index−1)上的 char 值在低代理项范围内，则(index−2)不小于 start，如果 char 数组中的(index−2)上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回(index−1)上的 char 值。

示例

本示例创建 char 数组，通过 Character 类的 codePointBefore()方法获取 char 数组中指定索引位置前面的代码点。并设置参数 start 的值，将返回值赋值给 int 类型变量。

```
char[] c1 = {'明','日','科','技'};
int cha0 = Character.codePointBefore (c1,1,0);
int cha1 = Character.codePointBefore (c1,2,1);
```


PDF电子书基地 http://dayo1982.400gb.com



方法声明 3

返回 CharSequence 的给定索引前面的代码点。

```
public static int codePointBefore(CharSequence seq, int index)
```

入口参数

seq：CharSequence 实例。

index：应该返回的代码点后面的索引。

返回值：返回 char 数组的给定索引前面的代码点。

抛出异常

NullPointerException：如果 seq 为 null。

IndexOutOfBoundsException：如果 index 参数小于 1 或者大于 seq.length()。

说明

如果 CharSequence 中的(index−1)上的 char 值在低代理项范围内，则(index−2)为非负，如果 CharSequence 中的(index−2)上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回(index−1)上的 char 值。

示例

本示例创建 CharSequence 实例，通过 Character 类的 codePointBefore()方法获取 CharSequence 实例中指定索引前面的代码点。将返回值赋值给 int 类型变量。

```
CharSequence seq = "明日科技";
int cha = Character.codePointBefore(seq, 2);
```

2.6.7 codePointCount 方法——返回字符数组的子数组中代码点的数量

返回 char 数组参数的子数组中 Unicode 代码点的数量。offset 参数是子数组的第一个 char 的索引，count 参数指定了 char 中的子数组的长度。子数组中不成对的代理项是按一个代码点算作一个项进行计数的。

方法声明 1

```
public static int codePointCount(char[] a, int offset, int count)
```

入口参数

a：char 数组。

offset：给定 char 数组中第一个 char 的索引。

count：char 中的子数组的长度。

返回值：返回 char 数组参数的子数组中 Unicode 代码点的数量。

抛出异常

NullPointerException：如果 a 为 null。

IndexOutOfBoundsException：如果 offset 或 count 为负，或者 offset+count 大于给定数组的长度。

示例

本示例创建 char 数组，通过 Character 类的 codePointCount()方法获取 char 数组参数的子数组中 Unicode 代码点的数量。并将返回值赋值给 int 类型变量 cha。

```
char[] c1 = {'明','日','科','技'};
int cha = Character.codePointCount(c1, 0, 3);
```

方法声明 2

返回指定字符序列的文本范围内的 Unicode 代码点数量。

```
public static int codePointCount(CharSequence seq, int beginIndex, int endIndex)
```

入口参数

seq：CharSequence 实例。

beginIndex：文本范围的第一个 char 的索引。

endIndex：文本范围的最后一个 char 后面的索引。

返回值：返回 char 数组参数的子数组中 Unicode 代码点的数量。

抛出异常

NullPointerException 如果 seq 为 null。

IndexOutOfBoundsException 如果 beginIndex 为负，或者 endIndex 大于给定序列的长度，或者 beginIndex 大于 endIndex。

说明

文本范围始于指定的 beginIndex，并扩展到索引 endIndex−1 上的 char。因此文本范围的长度（char 形式）为 endIndex-beginIndex。文本范围内的不成对代理项是按一个代码点算作一个项进行计数的。

示例

本示例创建 CharSequence 实例，通过 Character 类的 codePointBefore()方法获取 CharSequence 实例中指定索引前面的代码点。将返回值赋值给 int 类型变量。

```
CharSequence seq = "明日科技";
int cha = Character.codePointCount(seq, 0, 3);
```

2.6.8 compareTo 方法——比较字符对象

根据数字比较两个 Character 对象。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public int compareTo(Character anotherCharacter);
```

入口参数：anotherCharacter 为要比较的 Character。

返回值：如果 Character 等于 Character，则返回 0；如果该 Character 的数值小于参数 Character，则返回小于 0 的值；如果该 Character 的数值大于参数 Character，则返回大于 0 的值（有符号比较）。

注意

这是一次严格的数字比较；它并不依赖于区域。

示例

本示例创建 Character 对象 c1 和 c2，并通过 Character 类的 compare()方法判断对象 c1 和 c2 的大小。

```
Character c1 = new Character('2');
Character c2 = new Character('1');
int cha = c1.compareTo(c2);
```

2.6.9 equals 方法——判断字符对象相等

将此对象与指定对象比较。当且仅当参数不是 null，而是一个与此对象包含相同 char 值的 Character 对象时，结果才是 true。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：当且仅当参数不是 null，而是一个与此对象包含相同 char 值的 Character 对象时，结果才是 true。

示例

本示例创建 Character 和 String 对象，通过 equals 进行比较,将比较的结果赋给 b 并输出。当且仅当参数不是 null，而是一个与此对象包含相同 char 值的 Character 对象时，结果才是 true。所以 c.equals(s)为 false。

```
Character c = new Character('a');
String s = "a";
boolean b = c.equals(new Character('a'));
System.out.println(b);
b = c.equals(s);
System.out.println(b);
```

2.6.10 getNumericValue 方法——返回字符表示的 int 值

返回指定的 Unicode 字符表示的 int 值。

方法声明 1

```
public static int getNumericValue(char ch)
```

入口参数：ch 为要转换的字符。

返回值：返回指定的 Unicode 字符表示的 int 值。

说明

字母 A～Z 的大写（'\u0041'到'\u005A'）、小写（'\u0061'到'\u007A'）和全形参数（'\uFF21'到'\uFF3A'和'\uFF41'到'\uFF5A'）形式拥有 10～35 的数值。这独立于 Unicode 规范，该规范没有为这些 char 值分配数值。

如果字符中没有数值，则返回–1。如果字符中有一个数值，但无法将它表示为非负整数（例如，小数值），则返回–2。

此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 getNumericValue(int)方法。

示例

本示例将字符'\u216C'（罗马数字 50）返回一个值为 50 的整数。

```
char c = '\u216C';
int i = Character.getNumericValue(c);
System.out.println(i);
```

方法声明 2

返回指定的 Unicode 字符表示的 int 值。

```
public static int getNumericValue(int codePoint)
```

入口参数：codePoint 为要转换的字符（Unicode 代码点）。

返回值：以非负 int 值形式返回字符的数值；如果字符中有一个不是非负整数的数值，则返回–2；如果字符中没有数值，则返回–1。

说明

字母 A～Z 的大写（'\u0041'到'\u005A'）、小写（'\u0061'到'\u007A'）和全形参数（'\uFF21'到'\uFF3A'和'\uFF41'到'\uFF5A'）形式拥有 10～35 的数值。这独立于 Unicode 规范，该规范没有为这些 char 值分配数值。

如果字符中没有数字值，则返回–1。如果字符中有一个数值，但无法将它表示为非负整数（例如，小数值），则返回–2。

示例

本示例将字符'\u216C'（罗马数字 50）的数值返回一个值为 50 的整数。

```
int codePoint = '\u216C';
int i = Character.getNumericValue(codePoint);
System.out.println(i);
```


PDF电子书基地 http://dayo1982.400gb.com

2.6.11 getType 方法——返回一个指示字符的常规类别的值

返回一个指示字符的常规类别的值。

方法声明 1

```
public static int getType(char ch)
```

入口参数：ch 为要测试的字符。

返回值：表示字符的常规类别的 int 类型的值。

说明

此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 getType(int)方法。

示例

本示例返回了字符'a'的常规类别的值。

```
char ch = 'a';
System.out.println(Character.getType(ch));
```

方法声明 2

返回一个指示字符的常规类别的值。

```
public static int getType(int codePoint)
```

入口参数：codePoint 为要转换的字符（Unicode 代码点）。

返回值：表示字符的常规类别的 int 类型的值。

示例

本示例返回一个代码点的常规类别的值。

```
Character.getType(0x2F81A);
```

2.6.12 hashCode 方法——生成字符对象的哈希码

返回此 Character 的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回该 Character 对象的哈希码。

示例

本示例创建 1 个 Character 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Character c = new Character ('a');
int i =c.hashCode();
System.out.println(i);
```

2.6.13 isDefined 方法——判断是否为 Unicode 字符

确定字符是否被定义为 Unicode 中的字符。

方法声明 1

```
public static boolean isDefined(char ch)
```

入口参数：ch 为要测试的字符。

返回值：如果字符具有为 Unicode 中字符定义的含义，则返回 true；否则返回 false。

示例

本示例通过 isDefined()方法输出了字符是否被定义为 Unicode 中的字符的布尔值。

```
boolean b = Character.isDefined('a');
System.out.println(b);
```

方法声明 2

确定字符（Unicode 代码点）是否被定义为 Unicode 中的字符。

```
public static boolean isDefined(int codePoint)
```

入口参数：codePoint 为要测试的字符（Unicode 代码点）。

返回值：如果字符具有为 Unicode 中字符定义的含义，则返回 true；否则返回 false。

示例

本示例通过 isDefined()方法输出了代码点是否被定义为 Unicode 中的字符的布尔值。

```
boolean b = Character.isDefined(0x2F81A);
System.out.println(b);
```

2.6.14 isDigit 方法——判断是否为数字字符

检测指定字符是否为数字。如果字符为数字，则返回 true；否则返回 false。

方法声明 1

```
public static boolean isDigit(char ch);
```

入口参数：ch 为要检测的字符。

返回值：如果字符为数字，则返回 true；否则返回 false。

注意

此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isDigit(int)方法。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例通过 Character 类的 isDigit()方法判断 char 值是否是数字，并将结果赋给 boolean 变量。

```
char c2 ='明';
boolean cha = Character.isDigit(c2);
```

方法声明 2

判断指定字符（Unicode 代码点）是否为数字。如果通过 getType(codePoint)提供的字符的常规类别类型为 DECIMAL_DIGIT_NUMBER，则字符为数字。

```
public static boolean isDigit(int codePoint);
```

入口参数：codePoint 为要测试的字符（Unicode 代码点）。

返回值：如果字符为数字，则返回 true；否则返回 false。

示例

本示例应用 Character 类的 isDigit()方法，判断字符“明”的代码点 int 值 26126 是否是数字，并将结果赋给 boolean 变量。

```
int c2 =26126;
boolean cha = Character.isDigit(c2);
```

2.6.15 isLetter 方法——判断是否为字母字符

该方法用于判断参数字符是否为字母。

方法声明 1

```
public static boolean isLetter(char ch)
```

入口参数：ch 为要测试的字符。

返回值：如果字符为字母，则返回 true；否则返回 false。

示例

本示例通过 Character 类的 isLetter()方法，判断字符“*”是否为字母，并将结果赋值给 boolean 型变量 bool。由于“*”不属于字母范围内，因此 bool 为 false。

```
boolean c1 = Character.isLetter('*');
System.out.println(c1);
```

方法声明 2

该方法用于判断参数字符（Unicode 代码点）是否为字母。

```
public static boolean isLetter(int codePoint);
```

入口参数：codePoint 为要测试的字符（Unicode 代码点）。

返回值：如果字符为字母，则返回 true；否则返回 false。

示例

本示例通过 Character 类的 isLetter()方法，判断代码点 26126 是否为字母，并将结果赋值给 boolean 型变量 bool。

```
int c1 = 26126;
boolean cha = Character.isLetter(c1);
```

2.6.16 isLowerCase 方法——判断是否为小写字符

该方法用于判断指定字符是否为小写字母。

方法声明

```
public static boolean isLowerCase(char ch)
```

入口参数：ch 为要测试的字符。

返回值：如果字符为小写，则返回 true；否则返回 false。

示例

本示例通过 Character 类的 isLowerCase()方法，判断字符“L”是否为小写字母，并将结果赋值给 boolean 型变量 cha。

```
char c1 = 'L';
boolean cha = Character.isLowerCase(c1);
```

2.6.17 isUpperCase 方法——判断是否为大写字符

该方法用于判断指定字符是否为大写字母。

方法声明

```
public static boolean isUpperCase(char ch);
```

入口参数：ch 为要测试的字符。

返回值：如果字符为大写字母，则返回 true；否则返回 false。

示例

本示例通过 Character 类的 isUpperCase()方法，判断字符“L”是否为大写字母，并将结果赋值给 boolean 型变量 cha。

```
char c1 = 'L';
boolean cha = Character. isUpperCase(c1);
```

示例

本示例在项目中创建 Main 类，在主方法中创建 Character 对象，并判断变量的大小写状态。

```
public class Main{                                  //创建类
public static void main(String[] args) {            //定义主方法
    Character char1= new Character('M');            //声明 Character 对象
```


PDF电子书基地 http://dayo1982.400gb.com

```
        Character char2= new Character('n');                    //声明 Character 对象
        System.out.println("char1:"+"是大写字母吗？ "+char1.isUpperCase(char1));
        System.out.println("char2:"+"是小写字母吗？ "+char2.isLowerCase(char2));
    }
    }
```

2.6.18 toLowerCase 方法——转换为小写字符

使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为小写。

方法声明

```
public static char toLowerCase(char ch);
```

入口参数：ch 为要测试的字符。

返回值：等效于字符（Unicode 代码点）的小写形式，如果有的话；否则返回字符本身。

注意

通常使用 String.toLowerCase()将字符映射为小写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

示例

本示例通过 Character 类的 toLowerCase()方法，将字符“M”转换为小写字母，并将结果赋值给 char 型变量 cha。

```
char c1 = 'M';
char cha = Character.toLowerCase(c1);
```

2.6.19 toUpperCase 方法——转换为大写字符

使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为大写。

方法声明

```
public static char toUpperCase(char ch);
```

入口参数：ch 为要测试的字符。

返回值：等效于字符的大写形式；否则返回字符本身。

注意

通常使用 String.toUpperCase()将字符映射为大写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

示例

本示例通过 Character 类的 toUpperCase()方法，将字符“n”转换为大写字母，并将结果赋值给 char 型变量 cha。

```
char c1 = 'n';
char cha = Character.toUpperCase(c1);
```

2.7 Double——双精度数字类

java.lang 包中的 double 类在对象中包含一个基本类型 double 的值。该类提供多个方法，可以将 double 转换为 String，将 String 转换为 double，也提供了其他一些处理 double 时有用的常量和方法。

2.7.1 构造方法——Double 类的构造方法

Double 类的构造方法

构 造 方 法	使 用 说 明
Double(double value)	构造一个表示 double 参数的 Double 对象
Double(String str)	构造一个表示 Double 对象，表示用字符串表示的 double 类型的浮点值

2.7.2 常量——Double 类的常量

Double 类中包含多个常量，由于篇幅有限，下面列出部分常量，如下表所示。

Double 类的常量

常 量	含 义	数 值
MIN_VALUE	保存 double 类型的最小正非零值的常量	2^{-1074}
NaN	保存 double 类型的 NaN 值的常量	
SIZE	用于表示 double 值的位数	
TYPE	表示基本类型 double 的 Class 实例	

2.7.3 byteValue 方法——获取 byte 值

以 byte 形式返回此 Double 的值（通过强制转换为 byte）。

方法声明

```
public byte byteValue();
```

返回值：转换为 byte 类型的由此对象所表示的 double 值。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 Double 类对象 d1，通过 byteValue()方法将对象 d1 中的参数值强制转换为 byte 类型。

```
Double d1 = new Double(56.78);
byte dou = d1.byteValue();
```

2.7.4 compare 方法——比较双精度数字对象

比较两个指定的 Double 值。返回整数值的符号与以下调用返回整数的符号相同：new Double(d1).compareTo(new Double(d2))。

方法声明

```
public static int compare(double d1, double d2);
```

入口参数

d1：要比较的第一个 double 值。

d2：要比较的第二个 double 值。

返回值：返回值如果 d1 在数字上等于 d2，则返回 0；如果 d1 在数字上小于 d2，则返回小于 0 的值；如果 d1 在数字上大于 d2，则返回大于 0 的值。

示例

本示例通过 Double 类的 compare()方法比较 double 型变量 d1 和 d2 值的大小。

```
double d1 = 45.23;
double d2 = 45.22;
int dou = Double.compare(d1,d2);
```

2.7.5 compareTo 方法——比较两个 Double 对象

比较两个 Double 对象所表示的数值。

方法声明

```
public int compareTo(Double anotherDouble);
```

入口参数：anotherDouble 为要比较的 Double 值。

返回值：如果 anotherDouble 在数字上等于 Double，则返回 0；如果 anotherDouble 在数字上小于 Double，则返回小于 0 的值；如果 anotherDouble 在数字上大于 Double，则返回大于 0 的值。

示例

本示例创建 Double 对象 d1 和 d2，并通过 Double 类的 compareTo()方法比较对象 d1 和 d2 的大小。

```
Double d1 = new Double(45.23);
Double d2 = new Double(45.22);
int dou = d1.compareTo(d2);
```

2.7.6 intValue 方法——将此 Double 值以 int 形式返回

将此 Double 值以 int 形式返回（强制转换为 int 类型）。

方法声明

```
public int intValue();
```

返回值：转换为 int 类型的由此对象所表示的 double 值。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 intValue()方法将对象 d1 的参数强制转换为 int 型。

```
Double d1 = new Double(45.23F);
int dou = d1.intValue();
```

2.7.7 doubleToLongBits 方法——返回指定浮点值的表示形式

根据 IEEE754 浮点双精度格式("double format")位布局，返回指定浮点值的表示形式。

方法声明

```
public static long doubleToLongBits(double value)
```

入口参数：value 为双精度(double)浮点数。

返回值：表示浮点数的位。

说明

第 63 位（掩码 0x8000000000000000L 选定的位）表示浮点数的符号。第 62-52 位（掩码 0x7ff0000000000000L 选定的位）表示指数。第 51-0 位（掩码 0x000fffffffffffffL 选定的位）表示浮点数的有效数字（有时也称为尾数）。如果参数是正无穷大，则结果为 0x7ff0000000000000L。如果参数是负无穷大，则结果为 0xfff0000000000000L。如果参数是 NaN，则结果为 0x7ff8000000000000L。

在所有情况下，结果都是一个 long 整数，将其赋予 longBitsToDouble(long)方法将生成一个与 doubleToLongBits 的参数相同的浮点值（所有 NaN 值被压缩成一个“规范”NaN 值时除外）。

示例

本示例创通过 doubleToLongBits()方法将 double 数值转换为 IEEE754 浮点双精度格式("double format")位布局。

PDF电子书基地 http://dayo1982.400gb.com

```
double d = 123.456;
long l = Double.doubleToLongBits(d);
```

2.7.8 doubleToRawLongBits方法——保留NaN值返回指定浮点值的表示形式

根据 IEEE754 浮点“双精度格式”位布局，返回指定浮点值的表示形式，并保留 NaN 值。

方法声明

```
public static long doubleToRawLongBits(double value)
```

入口参数：value 为双精度(double)浮点数。

返回值：表示浮点数的位。

说明

第 63 位（掩码 0x8000000000000000L 选定的位）表示浮点数的符号。第 62-52 位（掩码 0x7ff0000000000000L 选定的位）表示指数。第 51-0 位（掩码 0x000fffffffffffffL 选定的位）表示浮点数的有效数字（有时也称为尾数）。如果参数是正无穷大，则结果为 0x7ff0000000000000L。如果参数是负无穷大，则结果为 0xfff0000000000000L。如果参数是 NaN，则结果是表示实际 NaN 值的 long 整数。与 doubleToLongBits 方法不同，doubleToRawLongBits 并没有压缩那些将 NaN 编码为一个“规范的”NaN 值的所有位模式。

在所有情况下，结果都是一个 long 整数，将其赋予 longBitsToDouble(long) 方法将生成一个与 doubleToRawLongBits 的参数相同的浮点值。

示例

本示例创通过 doubleToLongBits()方法将 double 数值转换为 IEEE754 浮点双精度格式("double format")位布局。

```
double d = 123.456;
long l = Double.doubleToRawLongBits(d);
```

2.7.9 doubleValue 方法——获取 double 值

返回此 Double 对象的 double 值。

方法声明

```
public double doubleValue();
```

返回值：此对象所表示的 double 值。

示例

本示例创建 Double 对象，并通过 doubleValue()方法将 Double 对象转换为

double 变量。并将转换后的结果赋给 double 变量 d2。

```
Double d1 = new Double(123.456);
double d2 = d1.doubleValue();
```

2.7.10 equals 方法——判断 Double 对象是否相等

将此对象与指定对象比较。当且仅当参数不是 null 而是 Double 对象，且表示的 Double 值与此对象表示的 double 值相同时，结果为 true。为此，当且仅当将方法 doubleToLongBits(double)应用于两个值所返回的 long 值相同时，才认为这两个 double 值相同。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

注意

在大多数情况下，对于 Double 类的两个实例 d1 和 d2，当且仅当 d1.doubleValue()==d2.doubleValue()为 true 时，d1.equals(d2)的值才为 true。但是，有以下两种例外情况:

如果 d1 和 d2 都表示 Double.NaN,那么即使 Double.NaN==Double.NaN 值为 false，equals 方法也将返回 true。

如果 d1 表示+0.0 而 d2 表示−0.0，或者相反，那么即使+0.0==−0.0 值为 true，equals 测试也将返回 false。

此定义使得哈希表得以正确操作。

示例

本示例创建 2 个 Double 对象，通过 equals 进行比较,将比较的结果赋给 b 并输出。

```
Double d1 = new Double(123.456);
Double d2 = new Double(123.456);
boolean b = d1.equals(d2);
System.out.println(b);
```

示例

本示例创建 2 个 Double 对象，通过 equals 进行比较,将比较的结果赋给 b 并输出。+0.0 和−0.0 比较结果为 false。

```
Double d1 = new Double(+0.0);
Double d2 = new Double(−0.0);
boolean b = d1.equals(d2);
System.out.println(b);
```


PDF电子书基地 http://dayo1982.400gb.com

2.7.11 floatValue 方法——获取 float 值

返回此 Double 对象的 float 值。

方法声明

```
public float floatValue();
```

返回值：转换为 float 类型的由此对象所表示的 double 值。

示例

本示例创建 Double 对象，并通过 floatValue()方法将 Double 对象转换为 float 变量。并将转换后的结果赋给 float 变量 f。

```
Double d = new Double(123.456);
float f = d.floatValue();
```

2.7.12 hashCode 方法——生成 Double 对象的哈希码

返回此 Double 对象的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回此 Double 对象的哈希码。

说明

结果是此 Double 对象所表示的基本 double 值的 long 整数位表示形式（与 doubleToLongBits(double)方法生成的结果完全一样）两部分整数之间的异或(XOR)。也就是说，哈希码就是以下表达式的值：

(int)(v^(v>>>32))

其中 v 的定义为：

```
long v=Double.doubleToLongBits(this.doubleValue());
```

示例

本示例创建 1 个 Double 对象，通过 hashCod()方法输出了 i 的哈希码。

```
Double d = new Double(123.456);
int i = d.hashCode();
```

2.7.13 isInfinite 方法——判断 Double 值的大小是否是无穷大

判断 Double 的值大小是无穷大，如果是则返回 true，否则返回 false。

方法声明 1

```
public boolean isInfinite();
```

返回值：如果是无穷大则返回 true，否则返回 false。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 isInfinite()方法判断对象 d1 的参数值是否是无穷大，并将结果赋给 boolean 型变量。

```
Double d1 = new Double(45.23F);
boolean dou = d1.isInfinite();
```

方法声明 2

判断如果指定数的数值是无穷大，则返回 true，否则返回 false。

```
public static boolean isInfinite(Double v);
```

入口参数：v 为指定要判断的 Double。

返回值：如果指定数的数值是无穷大，则返回 true，否则返回 false。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 isInfinite()方法判断对象 d1 的参数值是否是无穷大，并将结果赋给 boolean 型变量。

```
Double d1 = new Double(45.23F);
boolean dou= Double.isInfinite(d1);
```

2.7.14 isNaN 方法——判断 Double 值是否是一个非数字值

如果 Double 值是一个非数字(NaN)值，则返回 true，否则返回 false。

方法声明 1

```
public boolean isNaN();
```

返回值：如果 Double 值是一个非数字(NaN)值，则返回 true，否则返回 false。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 isNaN()方法判断对象 d1 的参数值是否是数字，并将结果赋给 boolean 型变量。

```
Double d1 = new Double(45.23F);
boolean dou = d1.isNaN();
```

方法声明 2

如果指定的数是一个 NaN 值，则返回 true；否则返回 false。

```
public static boolean isNaN(double v)
```

入口参数：v 为要判断的值。

返回值：如果参数值为 NaN，则返回 true；否则返回 false。

示例

本示例通过 isInfinite()方法判断 double 值是否为无穷大。

```
double d = 123.456;
boolean b = Double. isNaN (d);
```


PDF电子书基地 http://dayo1982.400gb.com

2.7.15 longBitsToDouble 方法——返回给定位表示形式的 double 值

返回对应于给定位表示形式的 double 值。根据 IEEE754 浮点“双精度格式”位布局，参数被视为浮点值表示形式。

方法声明

```
public static long doubleToLongBits(double value)
```

入口参数：bits 为任意 long 整数。

返回值：具有相同位模式的 double 浮点值。

说明

如果参数是 0x7ff0000000000000L，则结果为正无穷大。如果参数是 0xfff0000000000000L，则结果为负无穷大。如果参数值在 0x7ff0000000000001L～0x7fffffffffffffffL 之间或者在 0xfff0000000000001L～0xffffffffffffffffL 之间，则结果为 NaN。Java 提供的任何 IEEE754 浮点操作都不能区分具有不同位模式的两个同类型 NaN 值。不同的 NaN 值只能使用 Double.doubleToRawLongBits 方法区分。在所有其他情况下，设 s、e 和 m 为可以通过以下参数计算的三个值：

```
int s=((bits>>63)==0)?1: -1;
int e=(int)((bits>>52)&0x7ffL);
long m=(e==0)?(bits&0xfffffffffffffL)<<1: (bits&0xfffffffffffffL)| 0x10000000000000L;
```

那么浮点结果等于算术表达式 s · m · 2e–1075 的值。

注意，此方法不能返回与 long 参数具有完全相同位模式的 double NaN。IEEE754 区分了两种 NaN: quiet NaN 和 signaling NaN。这两种 NaN 之间的差别在 Java 中通常是不可见的。对 signaling NaN 进行的算术运算将它们转换为具有不同（但通常类似）位模式的 quiet NaN。但是在某些处理器上，只复制 signaling NaN 也执行这种转换。特别是在复制 signaling NaN 以将其返回给调用方法时，可能会执行这种转换。因此，longBitsToDouble 可能无法返回具有 signaling NaN 位模式的 double 值。所以，对于某些 long 值，doubleToRawLongBits(longBitsToDouble(start)) 可能不等于 start。此外，尽管所有 NaN 位模式（不管是 quiet NaN 还是 signaling NaN）都必须在上面提到的 NaN 范围内，但表示 signaling NaN 的特定位模式与平台有关。

示例

本示例创通过 doubleToLongBits()方法和 longBitsToDouble()方法演示了它们的相互转换。

```
double d1 = 123.456;
long l = Double.doubleToLongBits(d1);
```

```
System.out.println(l);
double d2 = Double.longBitsToDouble(l);
System.out.println(d2);
```

2.7.16 longValue 方法——获取 long 值

将 Double 值以 long 形式返回（强制转换为 long 类型）。

方法声明

```
public long longValue();
```

返回值：转换为 long 类型的由此对象所表示的 double 值。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 longValue()方法将对象 d1 的参数值强制转换成 long 型变量。

```
Double d1 = new Double(45.23F);
long dou = d1.longValue();
```

2.7.17 parseDouble 方法——将字符串解析为 double 值

获取指定 String 值的 double 表示形式。如果字符串中不包含可解析的 double 则抛出异常。该方法与 Double 类的 valueOf 方法一样。

方法声明

```
public static double parseDouble(String s)
        throws NumberFormatException;
```

入口参数：s 为指定要解析的字符串。

返回值：由字符串参数返回的 double 值。

抛出异常

NumberFormatException：如果字符串不包含可解析的 double 值。

示例

本示例创建 String 对象 d1，并通过 Double 类的 parseDouble()方法将对象 d1 的参数值解析成 double 型变量。

```
String d1 = new String("124");
double dou = Double.parseDouble(d1);
```

2.7.18 shortValue 方法——获取 short 值

将 Double 值以 short 形式返回（强制转换为 short）。

方法声明

```
public short shortValue();
```

返回值：转换为 short 类型由此对象所表示的 double 值。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 Double 对象 d1，并通过 Double 类的 shortValue()方法将对象 d1 的参数值强制转换成 short 型变量。

```
Double d1 = new Double(45.23F);
short dou = d1.shortValue();
```

2.7.19 toHexString 方法——生成双精度数字的十六进制字符串

获取 Double 参数的十六进制字符串表示形式。

方法声明

```
public static String toHexString(Double d);
```

入口参数：d 为要转换的 Double 值。

返回值：参数的十六进制字符串表示形式。

注意

下面提到的所有字符都是 ASCII 字符。

如果参数为 NaN，那么结果是字符串 “NaN”。否则，结果是表示参数的符号和数值（绝对值）的字符串。

如果符号为负，那么结果的第一个字符是'–'('\u002D')；如果符号为正，则结果中不显示符号字符。

如果 f 为无穷大，则用字符串 “Infinity” 表示；因此，正无穷大生成结果 “Infinity”，负无穷大生成结果 “–Infinity”。

如果 f 为 0，则用字符串 “0x0.0p0” 表示；因此，负 0 生成结果 “–0x0.0p0”，正 0 生成结果 “0x0.0p0”。

如果 f 是具有标准化表示形式的 Double 值，则使用子字符串表示有效位数和指数。有效位数用字符串 “0x1.” 表示，后跟该有效位数小数部分的小写十六进制表示形式。除非所有位数都为 0，否则移除十六进制表示形式中的尾部 0，在所有位数为 0 的情况下，可以用一个 0 表示。然后用 “p” 表示指数，后跟无偏指数的十进制字符串，该值与对指数值调用 Integer.toString 生成的值相同。

如果 f 是具有 subnormal 表示形式的 Double 值，则用字符 “0x0.” 表示有效位数，后跟该有效位数小数部分的十六进制表示形式。移除十六进制表示形式中的尾部 0。然后用 “p–126” 表示指数。注意，在 subnormal 有效位数中，至少必须有一个非 0 位数。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 toHexString()方法获取对象

d1 的参数值 0.25 的十六进制字符串表示形式。

```
Double d1 = new Double(0.25);
String dou = Double.toHexString(d1);
```

2.7.20 toString 方法——生成双精度数字的十进制字符串

返回 Double 参数的字符串表示形式。

方法声明 1

```
public static String toString(Double d);
```

入口参数：d 为要转换的 Double 参数。

返回值：参数的字符串表示形式。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 toString()方法获取对象 d1 的参数值的字符串表示形式。

```
Double d1 = new Double(0.25F);
String dou = Double.toString(d1);
```

方法声明 2

返回此 Double 对象的字符串表示形式。使用此对象表示的基本 double 值被转换为一个字符串，这与带一个参数的 toString 方法完全一样。

```
public String toString()
```

返回值：参数的字符串表示形式。

示例

本示例创建 Double 对象 d1，并通过 Double 类的 toString()方法获取对象 d1 的参数值的字符串表示形式。

```
Double d1 = new Double(0.25F);
String dou = d1.toString();
```

2.7.21 valueOf 方法——创建 Double 对象

返回表示指定的 Double 值的 Double 实例。如果不需要新的 Double 实例，则通常应优先使用此方法，而不是构造方法 Double(Double)，因为此方法可能通过缓存经常请求的值来显著提高空间和时间性能。

方法声明 1

```
public static Double valueOf(Double d);
```

入口参数：d 为一个指定的浮点值。

返回值：返回 d 的 double 实例。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 Double 对象 d1，并通过 Double 类的 toString()方法获取对象 d1 的参数值的字符串表示形式。

```
Double d1 = new Double(0.25F);
Double dou = Double.valueOf(d1);
```

方法声明 2

返回指定字符串的 Double 值的 Double 对象。如果字符串不包含可解析的数字，则抛出异常。

```
public static Double valueOf(String s)
        throws NumberFormatException;
```

入口参数：s 为一个指定要解析的 String。

返回值：保存用 String 参数表示的值的 Double 对象。

抛出异常

NumberFormatException：如果字符串不包含可解析的数字。

示例

本示例应用 Double 类的 valueOf()方法获取 String 变量 d1 的 Double 表示形式。

```
String d1 = "1.00001781796875001d";
Double dou = Double.valueOf(d1);
```

示例

本示例应用 Double 类的 valueOf()方法获取十六进制数字字符串表示形式“0x1.0p-2”的 Double 值。

```
String d1 = "0x1.0p-2";
Double dou = Double.valueOf(d1);
```

2.8 Float——浮点类

java.lang 包中的 Float 类在对象中包装了一个基本类型 Float 的值。该类中提供了几种方法，可将 Float 类型与 String 类型相互转换，还提供了处理 Float 类型时非常有用的其他一些常量和方法。

2.8.1 构造方法——Float 类的构造方法

Float 类的构造方法

构 造 方 法	使 用 说 明
Float(double value)	构造一个新分配的 Float 对象，表示转换为 float 类型的参数
Float(float value)	构造一个新分配的 Float 对象，表示基本的 float 参数
Float(String str)	构造一个新分配的 Float 对象，表示用字符串表示的 float 类型的浮点值

2.8.2 常量——Float类的常量

Float类中包含多个常量，由于篇幅有限，下面列出部分常量，如下表所示。

Float类的常量

常　量	含　义	数　值
MIN_NORMAL	保存float类型的最小正标准值的常量	2^{-126}
NaN	保存float类型的非数字（NaN）值的常量	
SIZE	表示一个float值所使用的位数	
MIN_VALUE	保存float类型数据的最小正非零值的常量	2^{-149}

2.8.3 byteValue方法——获取byte值

以byte形式返回此Float的值（通过强制转换为byte）。

方法声明

```
public byte byteValue();
```

返回值：转换为byte类型的由此对象所表示的Float值。

示例

本示例创建Float类对象f，通过byteValue()方法将对象f中的参数值强制转换为byte类型。

```
Float f = new Float(123.456f);
byte b = f.byteValue();
```

2.8.4 compare方法——比较Float对象

比较两个指定的 float 值。返回整数值的符号与以下调用返回整数的符号相同：new Float(f1).compareTo(new Float(f2))。

方法声明

```
public static int compare(float f1, float f2);
```

入口参数

f1：要比较的第一个float值。

f2：要比较的第二个float值。

返回值：返回值如果f1在数字上等于f2，则返回0；如果f1在数字上小于f2，则返回小于0的值；如果f1在数字上大于f2，则返回大于0的值。

示例

本示例通过Float类的compare()方法比较float型变量f1和f2值的大小。

```
float f1 = 456.23F;
float f2 = 456.22F;
int fol = Float.compare(f1,f2);
```


PDF电子书基地 http://dayo1982.400gb.com

2.8.5 compareTo 方法——比较两个 Float 对象所表示的数值

比较两个 Float 对象所表示的数值。

方法声明

```
public int compareTo(Float anotherFloat);
```

入口参数：anotherFloat 为要比较的 Float 值。

返回值：如果 anotherFloat 在数字上等于 Float，则返回 0；如果 anotherFloat 在数字上小于 Float，则返回小于 0 的值；如果 anotherFloat 在数字上大于 Float，则返回大于 0 的值。

示例

本示例创建 Float 对象 f1 和 f2，并通过 Float 类的 compareTo()方法比较对象 f1 和 f2 的大小。

```
Float f1 = new Float(456.23F);
Float f2 = new Float(456.22F);
int fol = f1.compareTo(f2);
```

2.8.6 doubleValue 方法——获取 double 值

返回此 Float 对象的 double 值。

方法声明

```
public double doubleValue()
```

返回值：此对象表示的 float 值被转换为 double 类型，并返回转换的结果。

示例

本示例创建 Float 类对象 f，通过 doubleValue()方法将对象 f 中的参数值强制转换为 double 类型。

```
Float f = new Float(123.456f);
double d = f.doubleValue();
```

2.8.7 equals 方法——判断 Double 对象相等

将此对象与指定对象进行比较。当且仅当参数不是 null 而是 Float 对象，且表示的 float 值与此对象表示的 float 值相同时，结果为 true。为此，当且仅当将方法 floatToLongBits(double)应用于两个值所返回的 int 值相同时，才认为这两个 float 值相同。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

注意

在大多数情况下，对于 Float 类的两个实例 f1 和 f2，当且仅当 f1.floatValue()==f2.floatValue()的值为 true 时，f1.equals(f2)的值才为 true。但是，有以下两种例外情况。

如果 f1 和 f2 都表示 Float.NaN，那么即使 Float.NaN==Float.NaN 的值为 false，equals 方法也将返回 true。

如果 f1 表示+0.0f，而 f2 表示−0.0f，或相反，那么即使 0.0f==−0.0f 的值为 true，equal 测试也将返回 false。

该定义使得哈希表得以正确操作。

示例

本示例创建两个 Float 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。

```
Float f1 = new Float(123.456f);
Float f2 = new Float(123.456f);
boolean b = f1.equals(f2);
System.out.println(b);
```

示例

本示例创建 2 个 Float 对象，通过 equals 进行比较，将比较的结果赋给 b 并输出。+0.0 和−0.0 比较结果为 false。

```
Float f1 = new Float(+0.0);
Float f2 = new Float(−0.0);
boolean b = f1.equals(f2);
System.out.println(b);
```

2.8.8 floatToIntBits 方法——返回浮点值的表示形式

根据 IEEE754 浮点“单一格式”位布局，返回指定浮点值的表示形式。

方法声明

```
public static int floatToIntBits(float value)
```

入口参数：value 为一个浮点数。

返回值：表示浮点数的位。

说明

第 31 位（掩码 0x80000000 选定的位）表示浮点数的符号。第 30～23 位（掩码 0x7f800000 选定的位）表示指数。第 22～0 位（掩码 0x007fffff 选定的位）表示浮点数的有效位数（有时也称为尾数）。

PDF电子书基地 http://dayo1982.400gb.com

如果参数为正无穷大，则结果为 0x7f800000。

如果参数为负无穷大，则结果为 0xff800000。

如果参数为 NaN，则结果为 0x7fc00000。

在所有情况下，结果都是一个整数，将其赋予 intBitsToFloat(int)方法将生成一个浮点值，该浮点值与 floatToIntBits 的参数相同（而所有 NaN 值则会生成一个“规范”NaN 值）。

示例

本示例创通过 floatToIntBits()方法将 float 数值转换为 IEEE754 浮点“单一格式”位布局。

```
float f = 123.456f;
int i = Float.floatToIntBits(f);
```

2.8.9 floatToRawIntBits 方法——保留非数字值返回指定浮点值的表示形式

根据 IEEE754 浮点“单一格式”位布局，返回指定浮点值的表示形式，并保留非数字(NaN)值。

方法声明

```
public static int floatToRawIntBits(float value)
```

入口参数：一个浮点数字。

返回值：表示浮点数的位。

说明

第 31 位（掩码 0x80000000 选定的位）表示浮点数的符号。第 30～23 位（掩码 0x7f800000 选定的位）表示指数。第 22～0 位（掩码 0x007fffff 选定的位）表示浮点数的有效位数（有时也称为尾数）。

如果参数为正无穷大，则结果为 0x7f800000。

如果参数为负无穷大，则结果为 0xff800000。

如果参数为 NaN，则结果是表示实际 NaN 值的整数。与 floatToIntBits 方法不同，floatToRawIntBits 不压缩所有将 NaN 编码为一个“规范”NaN 值的位模式。

在所有情况下，结果都是一个整数，将其赋予 intBitsToFloat(int)方法将生成一个与 floatToRawIntBits 的参数相同的浮点值。

示例

本示例创通过 floatToRawIntBits()方法将 float 数值根据 IEEE754 浮点“单一

格式”位布局，返回浮点值的表示形式，并保留非数字(NaN)值。

```
float f = 123.456f;
int i = Float.floatToRawIntBits(f);
```

2.8.10 floatValue 方法——获取 float 值

返回此 Float 对象的 float 值。

方法声明

```
public float floatValue()
```

返回值：此对象表示的 float 值。

示例

本示例创建 Float 类对象 f，通过 floatValue()方法将对象 Float 转换为基本类型 float。

```
Float f = new Float(123.456f);
float ff = f.floatValue();
```

2.8.11 hashCode 方法——返回 Float 对象的哈希码

返回 Float 对象的哈希码。

方法声明

```
public int hashCode()
```

返回值：返回此 Float 对象的哈希码。

说明

结果是此 Float 对象表示的基本 float 值的整数位表示形式，与 floatToIntBits(float)方法生成的结果完全一样。

示例

本示例创建一个 Float 对象 f，通过 hashCod()方法将 f 的哈希码赋值给 i。

```
Float f = new Float(123.456f);
int i = f.hashCode();
```

2.8.12 intBitsToFloat 方法——返回指定位表示形式的 float 值

返回对应于给定位表示形式的 float 值。根据 IEEE754 浮点“单一格式”位布局，该参数被视为浮点值表示形式。

方法声明

```
public static float intBitsToFloat(int bits)
```

入口参数：一个整数。

返回值：具有相同位模式的 float 浮点值。

PDF电子书基地 http://dayo1982.400gb.com

说明

如果参数值在 0x7f800001～0x7fffffff 或在 0xff800001～0xffffffff 之间，则结果为 NaN。Java 提供的任何 IEEE754 浮点操作都不能区分具有不同位模式的两个同类型 NaN 值。不同的 NaN 值只能使用 Float.floatToRawIntBits 方法区分。在所有其他情况下，设 s、e 和 m 为可以通过以下参数进行计算的 3 个值:

```
Nt s=((bits>>31)==0)?1: -1;
Int e=((bits>>23)&0xff);
Int m=(e==0)?(bits&0x7fffff)<<1:(bits&0x7fffff)|0x800000;
```

那么浮点结果等于算术表达式 s · m · 2e–150 的值。

注意，此方法不能返回与 int 参数具有完全相同位模式的 float NaN 值。IEEE754 区分了两种 NaN: quiet NaN 和 signaling NaN。这两种 NaN 之间的差别在 Java 中通常是不可见的。对 signaling NaN 进行的算术运算将它们转换为具有不同（但通常类似）位模式的 quiet NaN。但在某些只复制 signaling NaN 的处理器上也执行这种转换。特别是在复制 signaling NaN 以将其返回给调用方法时，可能会执行这种转换。因此，intBitsToFloat 可能无法返回具有 signaling NaN 位模式的 float 值。所以，对于某些 int 值，floatToRawIntBits(intBitsToFloat(start)) 可能不等于 start。此外，尽管所有 NaN 位模式（不管是 quiet NaN 还是 signaling NaN）都必须在前面确定的 NaN 范围内，但表示 signaling NaN 的特定位模式是与平台有关。

示例

本示例创通过 floatToIntBits()方法和 intBitsToFloat()方法演示了它们的相互转换。

```
float f = 123.456f;
int i = Float.floatToIntBits(f);
System.out.println(i);
float f2 = Float.intBitsToFloat(i);
System.out.println(f2);
```

2.8.13 intValue 方法——获取 int 值

将此 Float 值以 int 形式返回（强制转换为 int 类型）。

方法声明

```
public int intValue();
```

返回值：此对象表示的 float 值，该值被转换为 int 类型。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 intValue()方法将对象 f1 的参数

强制转换为 int 型。

```
Float f1 = new Float(456.23F);
int fol = f1.intValue();
```

2.8.14 isInfinite 方法——判断 float 值的大小是否是无穷大

判断 float 的值大小是无穷大，如果是则返回 true，否则返回 false。

方法声明 1

```
public boolean isInfinite();
```

返回值：如果此对象表示的值是正无穷大或负无穷大，则返回 true；否则返回 false。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 isInfinite()方法判断对象 f1 的参数值是否是无穷大，并将结果赋给 boolean 型变量。

```
Float f1 = new Float(456.23F);
boolean fol = f1.isInfinite();
```

方法声明 2

判断如果指定数的数值是无穷大，则返回 true，否则返回 false。

```
public static boolean isInfinite(float v);
```

入口参数：v 指定要判断的 float。

返回值：如果参数是正无穷大或负无穷大，则返回 true；否则返回 false。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 isInfinite()方法判断对象 f1 的参数值是否是无穷大，并将结果赋给 boolean 型变量。

```
Float f1 = new Float(456.23F);
boolean fol = Float.isInfinite(f1);
```

2.8.15 isNaN 方法——判断 Float 值是否是一个非数字值

如果 Float 值是一个非数字(NaN)值，则返回 true，否则返回 false。

方法声明 1

```
public boolean isNaN();
```

返回值：如果此对象表示的值是 NaN，则返回 true；否则返回 false。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 isNaN()方法判断对象 f1 的参数值是否是数字，并将结果赋给 boolean 型变量。

PDF电子书基地 http://dayo1982.400gb.com

```
Float f1 = new Float(456.23F);
boolean fol = f1.isNaN();
```

方法声明 2

判断如果 Float 值是一个非数字(NaN)值，则返回 true，否则返回 false。

```
public static boolean isNaN(float v)
```

返回值：如果此对象表示的值是 NaN，则返回 true；否则返回 false。

示例

本示例通过 Float 类的 isNaN()方法判断参数 float 的值是否是数字，并将结果赋给 boolean 型变量。

```
float f = 123.456f;
boolean b = Float.isNaN(f);
```

2.8.16 longValue 方法——获取 long 值

将 Float 值以 long 形式返回（强制转换为 long 类型）。

方法声明

```
public long longValue();
```

返回值：此对象表示的 float 值，该值被转换为 long 类型。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 longValue()方法将对象 f1 的参数值强制转换成 long 型变量。

```
Float f1 = new Float(456.23F);
long fol = f1.longValue();
```

2.8.17 parseFloat 方法——将字符串解析成 float 值

将指定的字符串解析成一个新的 float 值。返回值为用字符串参数表示的 float 值。

方法声明

```
public static float parseFloat(String s)
        throws NumberFormatException;
```

入口参数：s 为要解析的字符串。

返回值：用字符串参数表示的 float 值。

抛出异常

NumberFormatException：如果字符串不包含可解析的 float 值。

示例

本示例创建 String 对象 f1，并通过 Float 类的 parseFloat()方法将对象 f1 的参

数值解析成 float 型变量。

```
String f1 = new String("124");
float fo1 = Float.parseFloat(f1);
```

2.8.18 shortValue 方法——获取 short 值

将 Float 值以 short 形式返回（强制转换为 short）。

方法声明

```
public short shortValue();
```

返回值：此对象表示的 float 值，该值被转换为 short 类型。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 shortValue()方法将对象 f1 的参数值强制转换成 short 型变量。

```
Float f1 = new Float(456.23F);
short fo1 = f1.shortValue();
```

2.8.19 toHexString 方法——生成浮点数的十六进制字符串

获取 float 参数的十六进制字符串表示形式。

方法声明

```
public static String toHexString(float f);
```

入口参数：f 为要转换的 float 值。

返回值：参数的十六进制字符串表示形式。

注意

下面提到的所有字符都是 ASCII 字符。

如果参数为 NaN，那么结果是字符串 “NaN”。否则，结果是表示参数的符号和数值（绝对值）的字符串。

如果符号为负，那么结果的第一个字符是'-'('\u002D')；如果符号为正，则结果中不显示符号字符。

如果 f 为无穷大，则用字符串 “Infinity” 表示；因此，正无穷大生成结果 “Infinity”，负无穷大生成结果 “-Infinity”。

如果 f 为 0，则用字符串 “0x0.0p0” 表示；因此，负 0 生成结果 “-0x0.0p0”，正 0 生成结果 “0x0.0p0”。

如果 f 是具有标准化表示形式的 float 值，则使用子字符串表示有效位数和指数。有效位数用字符串 “0x1.” 表示，后跟该有效位数小数部分的小写十六进制表示形式。除非所有位数都为 0，否则移除十六进制表示形式中的尾部 0，在所有位数为 0 的情况下，可以用一个 0 表示。然后用 “p” 表示指

PDF电子书基地 http://dayo1982.400gb.com

数，后跟无偏指数的十进制字符串，该值与对指数值调用 Integer.toString 生成的值相同。

如果 f 是具有 subnormal 表示形式的 float 值，则用字符“0x0.”表示有效位数，后跟该有效位数小数部分的十六进制表示形式。移除十六进制表示形式中的尾部 0。然后用“p-126”表示指数。注意，在 subnormal 有效位数中，至少必须有一个非 0 位数。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 toHexString()方法获取对象 f1 的参数值 0.25 的十六进制字符串表示形式。

```
Float f1 = new Float(0.25);
String fol = Float.toHexString(f1);
```

2.8.20 toString 方法——生成浮点数的十进制字符串

返回 float 参数的字符串表示形式。

方法声明 1

```
public static String toString(float f);
```

入口参数：f 为要转换的 float 参数。

返回值：参数的字符串表示形式。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 toString()方法获取对象 f1 的参数值的字符串表示形式。

```
Float f1 = new Float(0.25F);
String fol = Float.toString(f1);
```

方法声明 2

返回此 Float 对象的字符串表示形式。使用此对象表示的基本 float 值被转换为一个 String，此方法与带一个参数的 toString 方法完全一样。

```
public String toString()
```

返回值：此对象的 String 表示。

示例

本示例创建 Float 对象 f，并通过 Float 类的 toString()方法获取对象 f 的值的字转换为符串表示形式。

```
Float f = new Float(123.456f);
String s = f.toString();
```

2.8.21 valueOf方法——创建浮点数对象

返回表示指定的 float 值的 Float 实例。如果不需要新的 Float 实例，则通常应优先使用此方法，而不是构造方法 Float(float)，因为此方法可能通过缓存经常请求的值来显著提高空间和时间性能。

方法声明 1

```
public static Float valueOf(float f);
```

入口参数：f 为一个指定的浮点值。

返回值：一个表示 f 的 Float 实例。

示例

本示例创建 Float 对象 f1，并通过 Float 类的 toString()方法获取对象 f1 的参数值的字符串表示形式。

```
Float f1 = new Float(0.25F);
float fo1 = Float.valueOf(f1);
```

方法声明 2

返回指定字符串的 float 值的 Float 对象。如果字符串不包含可解析的数字，则抛出异常。

```
public static Float valueOf(String s)
        throws NumberFormatException;
```

入口参数：s 为一个指定要解析的 String。

返回值：保存用 String 参数表示的值的 Float 对象。

抛出异常

NumberFormatException：如果字符串不包含可解析的数字。

示例

本示例应用 Float 类的 valueOf()方法获取 String 变量 f1 的 float 表示形式。

```
String f1 = "1.00017817968750001d";
float fo1 = Float.valueOf(f1);
```

示例

本示例应用 Float 类的 valueOf()方法获取十六进制数字字符串表示形式“0x1.0p-2”的 float 值。

```
String f1 = "0x1.0p-2";
float fo1 = Float.valueOf(f1);
```


PDF电子书基地 http://dayo1982.400gb.com

第3章 输入输出

Java 由数据流处理输入输出模式。输入模式是由程序创建某个信息来源的数据流并打开该数据流获取指定信息来源的数据，数据来源可以是文件、网络、压缩包或者其他数据源。输出模式是由程序创建某个输出对象的数据流，并打开该数据对象，将数据写入数据流。数据流对象会将数据存储到相应的目标中。这些输出目标可以是文件、网络、压缩包、控制台或其他数据输出目标。

3.1 BufferedInputStream 类——缓冲字节输入流

java.io 包中的 BufferedInputStream 类是缓冲输入流，即为输入流添加了缓冲区功能。

3.1.1 构造方法——BufferedInputStream 类的构造方法

BufferedInputStream 类的构造方法如下表所示。

BufferedInputStream 类的构造方法

构 造 方 法	使 用 说 明
BufferedInputStream(InputStream in)	创建一个 BufferedInputStream 并保存其参数
BufferedInputStream(InputStream in, int size)	创建具有指定缓存区大小的 BufferedInputStream 并保存其参数

3.1.2 available 方法——返回可以读取的估计字节数

返回可以从此输入流读取（或跳过）、且不受此输入流接下来的方法调用阻塞的估计字节数。接下来的调用可能是同一个线程，也可能是不同的线程。一次读取或跳过这么多字节将不会受阻塞，但可以读取或跳过数量更少的字节。

此方法返回缓冲区中剩余的待读取字节数(count-pos)与调用 in.available()的结果之和。

方法声明

```
public int available() throws IOException
```

返回值：可以不受阻塞地从此输入流读取（或跳过）的估计字节数。

抛出异常：IOException，如果此文件输入流已通过调用 close 关闭，或者发生 I/O 错误。

示例

本示例创建文件输入流对象，并通过 available()方法估计了剩余的字节数，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H:\\Getif.java");        // 用文件路径新建一个文件对象
    InputStream in = new FileInputStream(file);
    BufferedInputStream bis = new BufferedInputStream(in);
    System.out.println("文件大小为" + file.length() + "Bytes");
    bis.read();        //读取一个字节
    System.out.println("估计文件还有" + bis.available() + "Bytes");
    bis.close();
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.1.3 close 方法——关闭此输入流并释放所有关联的系统资源

关闭此输入流并释放与该流关联的所有系统资源。再调用 read()、skip()等方法，会抛出 IOException 异常。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例实现创建缓冲输入流，读取指定磁盘文件。通过调用 close()方法将流关闭。

```
try{
File file = new File("H://Getif.java");
FileInputStream fin = new FileInputStream(file);
BufferedInputStream bip = new BufferedInputStream(fin);
byte[] by = new byte[10];
while(bip.read(by) != -1){
    System.out.println(new String(by));
}
bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.1.4 mark 方法——记录当前指针所在的位置

在此输入流中记录当前指针所在的位置。

方法声明

```
public void mark(int readlimit)
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：readlimit 是告知此输入流在标记位置失效之前允许读取的字节数。

示例

本示例实现创建缓冲输入流，实现循环读取指定磁盘文件中前 5 个字节。

```
try {
    File file = new File("H://Getif.java");
    FileInputStream fin = new FileInputStream(file);
    BufferedInputStream bip = new BufferedInputStream(fin);
    int count = 0;
    bip.mark(50);
    for (int i = 0; i < file.length(); i++) {
        count++;
        int read = bip.read();
        if (count % 5 == 0)
            bip.reset();
        System.out.println((char) read);
        }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.1.5 markSupported 方法——测试是否支持 mark()方法和 reset()方法

该方法用于测试输入流是否支持 mark()方法和 reset()方法。

方法声明

```
public boolean markSupported()
```

返回值：如果支持 mark()和 reset()方法则返回 true，否则返回 false。

示例

本示例分别创建 FileInputStream 类与 BufferedInputStream 类，通过 markSupported()方法判断这两个类是否支持 mark()与 reset()方法，并给出相应的提示信息。

```
try {
    File file = new File("H://Getif.java");
    FileInputStream fin = new FileInputStream(file);
    if (fin.markSupported()) {
        System.out.println("FileInputStream 类支持 mark()与 reset()方法");
    }
    BufferedInputStream bip = new BufferedInputStream(fin);
    if (bip.markSupported()) {
        System.out.println("BufferedInputStream 类支持 mark()与 reset()方法");
    }
} catch (Exception e) {
```

```
        e.printStackTrace();
    }
```

3.1.6 read 方法——从输入流中读取数据

此方法用于从输入流中读取数据。该方法用于提供了多种重载形式。

方法声明 1

```
public abstract int read() throws IOException
```

返回值：从输入流中读取数据的下一个字节。返回 0～255 范围内的 int 字节值。如果到达流的末尾，则返回-1。

抛出异常：IOException。

示例

本示例创建缓冲输入流，通过 read()方法实现从输入流中每次读取一个字节，通过调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    FileInputStream fin = new FileInputStream(file);
    BufferedInputStream bip = new BufferedInputStream(fin);
    int length;
    for(int i =0;i<file.length();i++){
        length = bip.read();
        System.out.print((char)length);
    }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 2

```
public int read(byte[] b,int off,int len) throws IOException
```

读取 len 个字节，放置到以下标 off 开始的字节数组 b 中。

入口参数

b：保存读取数据的 byte 数组。

off：指定 read()方法把数据存放在字节数组 b 中偏移量。

len：执行该方法将读取的最大字节数。

返回值：返回实际读取的字节个数。

抛出异常：IOException。

示例

本示例创建缓冲输入流，通过 read()方法给定参数来限制读取流中数据的字节数。

PDF电子书基地 http://dayo1982.400gb.com

```
try {
    File file = new File("H://Getif.java");
    FileInputStream fin = new FileInputStream(file);
    BufferedInputStream bip = new BufferedInputStream(fin);
    byte[] bs = new byte[20];
    int i = bip.read(bs, 0, 10);
    System.out.println("控制台输入的内容：" + new String(bs).trim());
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.1.7 reset 方法——重新定位输入流

将输入流重新定位到最后一次对此输入流调用 mark()方法时的位置。

方法声明

```
public void reset() throws IOException
```

抛出异常：IOException。

示例

本示例分别创建 BufferedInputStream 类对象，通过调用 read()方法读取流中数据。并通过 reset()方法实现当指定变量等于 5 时，重新读取文件中数据。

```
try {
    File file = new File("H://Getif.java");
    FileInputStream fin = new FileInputStream(file);
    BufferedInputStream bip = new BufferedInputStream(fin);
    int count = 0;
    bip.mark(50);
    for (int i = 0; i < file.length(); i++) {
        count++;
        int read = bip.read();
        if (count == 5)
            bip.reset();
        System.out.println((char) read);
    }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.1.8 skip 方法——跳过并丢弃指定字节数量的数据

从输入流中跳过并丢弃 *n* 个字节的数据。

方法声明

```
public long skip(long n) throws IOException
```

返回值：实际跳过的字节数。

抛出异常：IOException，如果 n 为负，如果该流不支持查找操作，或者发生 I/O 错误，抛出 IOException。

说明

出于各种原因，skip 方法最终跳过的字节数可能更少一些，甚至可能为 0。如果 n 为负，则抛出 IOException，即使 InputStream 超类的 skip 方法在这种情况下没有执行任何操作，返回实际跳过的字节数。此方法跳过的字节可能多于底层实现文件中剩余的字节。这不会产生异常，并且跳过的字节数可能包括底层实现文件的 EOF（文件结束符）之后的一些字节数。如果试图在跳过末尾之后读取流，那么会返回-1（指示文件结束）。

示例

本示例创建文件输入流对象，并通过 skip()方法跳过 10 个字节，然后开始读数据并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H:\\Getif.java");          // 用文件路径新建一个文件对象
    InputStream in = new FileInputStream(file);
    BufferedInputStream bis = new BufferedInputStream(in);
    bis.skip(10);
    int i;
    while((i = bis.read()) != -1){
        System.out.print((char)i);
    }
    bis.close();
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.2 BufferedOutputStream 类——缓冲字节输出流

java.io 包中 BufferedOutputStream 类，实现缓冲的输出流。通过缓冲输出流可以将各个字节写入底层输出流中，而不必针对每次字节写入调用底层系统。

3.2.1 构造方法——BufferedOutputStream 类的构造方法

BufferedOutputStream 类的构造方法如下表所示。

BufferedOutputStream 类的构造方法

构 造 方 法	使 用 说 明
BufferedOutputStream(OutputStream out)	创建一个新的缓冲输出流，以将数据写入指定的底层输出流
BufferedOutputStream(OutputStream out, int size)	创建一个新的缓冲输出流，以将具有指定缓冲区大小的数据写入指定的底层输出流

PDF电子书基地 http://dayo1982.400gb.com

3.2.2 flush 方法——刷新此缓冲输出流

刷新此缓冲输出流，并输出所有被缓存的字节。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException。

示例

本示例创建 BufferedOutputStream 类对象，通过 write()方法向流中写数据，并通过 flush()方法刷新缓存。

```
try {
    File file = new File("H://Getif.java");
    FileOutputStream fos = new FileOutputStream(file);
    BufferedOutputStream bus = new BufferedOutputStream(fos);
    byte[] by = "香蕉".getBytes();
    bus.write(by);
    byte[] by2 = "苹果".getBytes();
    bus.write(by2);
    byte[] by3 = "橘子".getBytes();
    bus.write(by3);
    bus.flush();
    bus.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.2.3 write 方法——向输出流中写数据

该方法实现向输出流中写数据，该类提供了多种重载形式。

方法声明 1

通过在 write()方法中给定参数，实现向输出流中写入指定参数字节数据。

```
public void write(int b) throws IOException
```

入口参数：b 是要写入的字节数。

抛出异常：IOException。

示例

本示例创建 BufferedOutputStream 类对象，通过调用 write()方法实现向输出流中写数据。

```
try {
    File file = new File("H://Getif.java");
    FileOutputStream fos = new FileOutputStream(file);
    BufferedOutputStream bus = new BufferedOutputStream(fos);
    bus.write(97);
    bus.close();
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
```

方法声明 2

本方法实现将数组 b 中从 off 下标开始的数据写入到文件中，写入文件的长度为指定的参数 length。

```
public void write(byte[] b,int off,int len) throws IOException
```

入口参数

b：要写入流中的数组。

off：要写入流的数组开始索引位置。

len：要写入流的字节数。

抛出异常：IOException。

示例

本示例创建 BufferedOutputStream 类对象，通过调用 write()方法实现向输出流中写指定长度的数据。

```
try {
    File file = new File("H://Getif.java");
    FileOutputStream fos = new FileOutputStream(file);
    BufferedOutputStream bus = new BufferedOutputStream(fos);
    byte b[] = "使用 BufferedOutputStream 类向流中写数据".getBytes();
    bus.write(b, 0,10);
    bus.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

示例

本示例通过缓冲输入流读取当前 H 盘中的 Getif.java 文件，并通过缓存输出流将读取的数据保存到 H 盘的 12.txt 文件中。

```
File file = new File("H://Getif.java");
File fileOut = new File("H://12.txt");
try {
    if(!fileOut.exists()){
        fileOut.createNewFile();
    }
FileOutputStream fon = new FileOutputStream(fileOut);
FileInputStream fin = new FileInputStream(file);
byte[] by = new byte[1];
BufferedInputStream bip = new BufferedInputStream(fin);
BufferedOutputStream bop = new BufferedOutputStream(fon);
while(bip.read(by) != -1){
    bop.write(by);
```


PDF电子书基地 http://dayo1982.400gb.com

```
        }
        bop.flush();
        fon.close();
        bip.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.3 BufferedReader 类——缓冲字符输入流

java.io 包中的 BufferedReader 类，实现从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取。

3.3.1 构造方法——BufferedReader 类的构造方法

BufferedReader 类的构造方法如下表所示。

BufferedReader 类的构造方法

构 造 方 法	使 用 说 明
BufferedReader(Reader rd)	创建一个使用默认大小输入缓冲区的缓冲字符输入流
BufferedReader(Reader rd, int size)	创建一个使用指定大小输入缓冲区的缓冲字符输入流

3.3.2 close 方法——关闭流并释放与之关联的所有资源

该方法关闭流并释放与之关联的所有资源。调用该方法关闭流后，再调用 read()、mark()、skip()方法将会抛出 IOException 异常。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例通过缓冲字符输入流，将控制台中输入的信息重新输出到控制台上，通过调用 close()方法将流关闭。

```
BufferedReader buf;
String str =null;
buf=new BufferedReader(new InputStreamReader(System.in));
System.out.println("输入一个字符串:");
try {
    str=buf.readLine();
    buf.close();
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println(str);
```

3.3.3 mark 方法——记录当前读指针所在的位置

在流中记录当前读指针所在的位置。

方法声明

```
public void mark(int readLimit) throws IOException
```

入口参数：readLimit 表示指针读出 readLimit 个字节后所标记的指针位置才失效。

抛出异常：IOException。

示例

本示例通过缓冲字符输入流，读取文件中数据，并通过 mark()方法在流中做标记，实现反复将指定文件中前 5 个字符输出。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
    BufferedReader bip = new BufferedReader(fin);
    int count = 0;
    bip.mark(50);
    for (int i = 0; i < file.length(); i++) {
        count++;
        int read = bip.read();
        if (count % 5 == 0)
            bip.reset();
        System.out.println((char) read);
        }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.3.4 markSupported 方法——判断指定流是否支持 mark()操作

判断指定流是否支持 mark()操作，并将判断结果返回。

方法声明

```
public boolean markSupported()
```

返回值：如果该流支持 mark()方法则返回 true，否则返回 false。

示例

本示例创建缓冲输入流对象，通过 markSupported()方法判断该缓冲输入流是否支持 mark()方法，并给出相应的提示信息。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
```


PDF电子书基地 http://dayo1982.400gb.com

```
        BufferedReader bip = new BufferedReader(fin);
        if(bip.markSupported()){
            System.out.println("BufferedReader 支持 mark()方法");
        }
        else{
            System.out.println("BufferedReader 不支持 mark()方法");
        }
        bip.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.3.5 read 方法——从缓冲输入流中读取字符

该方法用于从缓冲输入流中读取字符，read()方法提供了多种重载形式。

方法声明 1

```
public int read() throws IOException
```

该方法从缓冲区中读取单个字符。

抛出异常：IOException。

示例

本示例创建缓冲输入流对象，读取指定文件中数据。通过调用 read()方法实现从流中读取一个字节元素，并将其输出在控制台中。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
    BufferedReader bip = new BufferedReader(fin);
    int read;
    while ((read = bip.read()) != -1) {
        char ch = (char) read;
        System.out.print(ch);
    }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 2

该方法从流中读取指定参数的字符，将其存放在以下标 off 开始的字符数组中。

```
public int read(char[] cbuf,int off,int len) throws IOException
```

入口参数

cbhf：保存读取结果的 char 数组。

off：将读取数据放置在下标 off 开始的字符数组中。

len：读取的字符数。

返回值：返回读取的字符数，若到达流的末尾则返回-1。

抛出异常：IOException。

示例

本示例创建缓冲字符输入流对象，并通过 read()方法读取指定文件中前 10 个字符元素，将其保存在字符数组 b 中，并将保存读取结果的字符数组 b 输出。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
    BufferedReader bip = new BufferedReader(fin);
    int read;
    char[] b = new char[10];
    bip.read(b,0,10);
    System.out.println(new String(b));
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.3.6 readLine 方法——读取一个文本行

该方法读取一个文本行，并将其返回为字符串，若无数据可读，则返回 null。

方法声明

```
public String readLine() throws IOException
```

返回值：返回包含读取行内容的字符串，如果到达流末尾，则返回 null。

抛出异常：IOException。

示例

本示例创建缓冲输入流对象，并通过 readLine()方法实现分行读取指定文件中数据，并将其分行显示在控制台上。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
    BufferedReader bip = new BufferedReader(fin);
    String s = null;
    int i = 0;
    while ((s = bip.readLine()) != null) {
        i++;
        System.out.println("第" + i + "行:" + s);
    }
    bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.3.7 ready 方法——判断此流是否已准备好被读取

判断此流是否已准备好被读取。如果缓冲区不为空，或者底层字符流已准备就绪，则缓冲的字符流准备就绪。

方法声明

```
public boolean ready() throws IOException
```

返回值：如果保证下一个 read()不阻塞输入，则返回 true，否则返回 false。注意，返回 false 并不保证阻塞下一次读取。

抛出异常：如果发生 I/O 错误，抛出 IOException。

示例

本示例创建字符输入流对象，并通过 ready()方法判断是否已准备好被读取，如果为 true 则开始读取一行数据，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H:\\Getif.java");        // 用文件路径新建一个文件对象
    FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr);
    if(br.ready()){ //判断是否已准备好被读取
        System.out.println(br.readLine());  //读取一行数据
    }
    br.close();
    fr.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.3.8 reset 方法——重定位指针

把指针重新指向用 mark()方法所记录的位置。

方法声明

如果从未标记过该输入流，或者标记已经失效，该方法抛出 IOException 异常。

```
public void reset() throws IOException
```

抛出异常：IOException。

示例

本示例创建缓冲输入流对象，并通过 mark()方法向流中添加标记，通过 reset()方法实现反复读取指定文件中前 3 个字符。

```
try {
    File file = new File("H://Getif.java");
    FileReader fin = new FileReader(file);
    BufferedReader bip = new BufferedReader(fin);
    int curr = 0;
```

```
bip.mark(10);
for (int i = 0; i < file.length(); i++) {
    curr++;
    int read = bip.read();
    if (curr % 3 == 0)
        bip.reset();
    System.out.println((char) read);
}
bip.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.3.9 skip 方法——跳过指定数量的字符

跳过指定数量的字符。

方法声明

```
public long skip(long n) throws IOException
```

返回值：实际跳过的字符数。

抛出异常

IllegalArgumentException：如果 n 为负。

IOException：如果发生 I/O 错误。

示例

本示例创建通过 skip()方法跳过 10 个字符，然后开始读数据并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H:\\Getif.java");        // 用文件路径新建一个文件对象
    FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr);
    br.skip(10);      //跳过 10 个字符
    int i;
    while ((i = br.read()) != -1) {
        System.out.print((char) i);
    }
    br.close();
    fr.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.4 BufferedWriter 类——缓冲字符输出流

java.io 包中的 BufferedWriter 类，实现文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入。

PDF电子书基地 http://dayo1982.400gb.com

3.4.1 构造方法——BufferedWriter 类的构造方法

该类构造方法如下表所示。

BufferedWriter 类的构造方法

构 造 方 法	使 用 说 明
BufferedWriter(Writer out)	创建一个使用默认大小输入缓冲区的缓冲字符输出流
BufferedWriter(Writer out, int size)	创建一个使用指定大小输出缓冲区的新缓冲字符输出流

3.4.2 close 方法——关闭流

该方法关闭流，在调用 close()方法后调用 write()或 flush()方法将导致 IOException 异常。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例通过缓冲输出流实现向 H 盘的 12.txt 文件中写数据，通过调用 close()方法将流关闭。

```
File file = new File("H:/12.txt");
    try {
        if (!file.exists()) {
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file);
        BufferedWriter bufw = new BufferedWriter(fw);
        String str1 = "最近好吗?";
        bufw.write(str1);
        bufw.close();
        fw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.4.3 flush 方法——刷新缓冲区

该方法用于刷新缓冲区，并输出所有被缓存的字节。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException。

示例

本示例通过缓冲输出流实现向 H 盘的 12.txt 文件中写入字符串数组中的各元

素，并通过 flush()方法刷新缓冲区。

```
File file = new File("H:/12.txt");
try {
    if (!file.exists()) {
        file.createNewFile();
    }
    String strArr[] = { "Java 编程词典", "Java 初学者", "Java 工程师" };
    FileWriter fw = new FileWriter(file);
    BufferedWriter bufw = new BufferedWriter(fw);
    for (int i = 0; i < strArr.length; i++) {
        bufw.write(strArr[i]);
    }
    bufw.flush();
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.4.4 newLine 方法——写入一个行分隔符

该方法写入一个行分隔符，行分隔符字符串由系统属性 line.separator 定义，并且不一定是单个新行。

方法声明

```
public void newLine() throws IOException
```

抛出异常：IOException。

示例

本示例通过缓冲输出流实现向 H 盘的 12.txt 文件中写数据，并通过 newLine()方法实现将数组中的元素分行写到文本文件中。

```
String content[] = { "好久不见" , "最近好吗" , "常联系" };
File file = new File("H:/12.txt");
try {
    if(!file.exists()){
        file.createNewFile();
    }
    FileWriter fw = new FileWriter(file);
    BufferedWriter bufw = new BufferedWriter(fw);
    for (int k = 0; k < content.length; k++) {
        bufw.write(content[k]);
        bufw.newLine();
    }
    bufw.close();
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.4.5 write 方法——向缓冲区中写数据

该方法实现向缓冲区中写数据，该方法提供了多种重载形式。

方法声明 1

该方法实现写入字符数组中的某一部分，从数组 cbuf 中下标 off 开始，长度 len 的字节写入流中。

```
public void write(char[] cbuf,int off,int len) throws IOException
```

入口参数

cbuf：要写入输出流的字符数组。

off：开始写入流的字符数组的索引位置。

len：写入流的字符数组的长度。

抛出异常：IOException。

示例

本示例创建缓冲输出流对象，并创建字符数组，通过 read()方法将字符数组中索引位置在 0 的元素开始向输出流中写数据，写入的长度是 5。

```
File file = new File("H:/12.txt");
try {
    if (!file.exists()) {
        file.createNewFile();
    }
    FileWriter fw = new FileWriter(file);
    BufferedWriter bufw = new BufferedWriter(fw);
    char[] ch = { 'a', 'b', 'c', 'd', 'e' };
    bufw.write(ch, 0, 5);
    bufw.flush();
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 2

该方法实现向输出流中写入单个字符。

```
public void write(int i) throws IOException
```

入口参数

i：要写入流的字节。

抛出异常：IOException。

示例

本示例创建缓冲输出流对象，通过 write()方法向指定文件中写数据。由于数字 97 对应的 ASCII 值是小写字母“a”，因此指定文件中写入的是小写字母“a”。

```
File file = new File("H:/12.txt");
try {
    if (!file.exists()) {
        file.createNewFile();
    }
    FileWriter fw = new FileWriter(file);
    BufferedWriter bufw = new BufferedWriter(fw);
    bufw.write(97);
    bufw.flush();
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 3

该方法向流中写入指定参数字符串，同时给定写入流的字符串的开始下标位置，以及写入流的字符长度。

```
public void write(String str,int off,int len) throws IOException
```

入口参数

str：要写入流的字符串对象。

off：指定要写入流的字符串的开始下标。

len：将长度为 len 的字符写入流。

抛出异常：IOException。

示例

本示例创建缓冲输出流对象，并创建字符串。通过 write()方法将字符串中某部分写入指定磁盘文件中。

```
File file = new File("H:/12.txt");
try {
    if (!file.exists()) {
        file.createNewFile();
    }
    FileWriter fw = new FileWriter(file);
    BufferedWriter bufw = new BufferedWriter(fw);
    String str = "Java 编程词典 编程之家";
    bufw.write(str,0,12);
    bufw.flush();
    fw.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.5 DataInputStream 类——数据输入流

java.io 包中的 DataInputStream 类是数据输入流，允许应用程序以与机器无

PDF电子书基地 http://dayo1982.400gb.com

关方式从底层输入流中读取基本 Java 数据类型。也就是说，当读取一个数据时，不必再关心这个数值应当是什么字节。

该类的构造方法语法如下：

```
DataInputStream(InputStream in)
```

3.5.1 read 方法——读取一定数量的字节

从包含的输入流中读取一定数量的字节，并将它们存储到缓冲区数组 b 中。以整数形式返回实际读取的字节数。在输入数据可用、检测到文件末尾（end of file）或抛出异常之前，此方法将一直阻塞。

方法声明 1

```
public final int read(byte[] b) throws IOException
```

返回值：读入缓冲区的字节总数。如果因为已经到达流末尾而没有更多的数据，则返回-1。

抛出异常：IOException，如果不是因为流位于文件末尾而无法读取第一个字节；该流已关闭并且底层输入流在关闭后不支持读取操作；发生其他 I/O 错误。

说明

如果 b 为 null，则抛出 NullPointerException。如果 b 的长度为 0，则不读取字节并返回 0；否则，尝试读取至少一个字节。如果因为流位于文件末尾而没有字节可用，则返回值-1；否则至少读取一个字节并将其存储到 b 中。将读取的第一个字节存储到元素 b[0]中，将下一个字节存储到 b[1]中，依此类推。读取的字节数至多等于 b 的长度。设 k 为实际读取的字节数；这些字节将存储在从 b[0]～b[k-1]的元素中，b[k]～b[b.length-1]的元素不受影响。

示例

本示例创建了一个文件输入流、一个数据输入流，通过 DataInputStream 类的 read()方法将指定磁盘文件读取到字节数组 b 中并输出。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    //创建文件字节流
    FileInputStream fis = new FileInputStream(file);
    //创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    byte[] b = new byte[50];
    dis.read(b);      //将数据读到字节数组 b 中
    for (int i = 0; i < b.length; i++) {
        System.out.println(b[i]);
    }
    dis.close();
```

```
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

方法声明 2

从包含的输入流中将最多 len 个字节读入一个 byte 数组中。尽量读取 len 个字节，但读取的字节数可能少于 len 个，也可能为零。以整数形式返回实际读取的字节数。在输入数据可用、检测到文件末尾或抛出异常之前，此方法将阻塞。

```
public static File createTempFile(String prefix, String suffix, File directory)
        throws IOException
```

入口参数

b：存储读取数据的缓冲区。

off：目标数组 b 中的起始偏移量。

len：读取的最大字节数。

返回值：读入缓冲区的字节总数。如果因为已经到达流末尾而没有更多的数据，则返回-1。

抛出异常：IOException，如果不是因为流位于文件末尾而无法读取第一个字节；该流已关闭并且底层输入流在关闭后不支持读取操作；发生其他 I/O 错误。

说明

如果 len 为 0，则不读取任何字节并返回 0；否则，尝试读取至少一个字节。如果因为流位于文件末尾而没有字节可用，则返回值-1；否则，至少读取一个字节并将其存储到 b 中。将读取的第一个字节存储到元素 b[off]中，将下一个字节存储到 b[off+1]中，依此类推。读取的字节数至多等于 len。设 k 为实际读取的字节数；这些字节将存储在 b[off]～b[off+k-1]的元素中，b[off+k]～b[off+len-1]的元素不受影响。在所有情况下，b[0]～b[off]的元素和 b[off+len]～b[b.length-1]的元素都不受影响。

示例

本示例创建了一个文件输入流一个数据输入流，通过 DataInputStream 类的 read()方法将指定磁盘文件读取到字节数组 b 中（起始偏移量为 2，读取 10 个字节）并输出。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    //创建文件字节流
    FileInputStream fis = new FileInputStream(file);
    //创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
```


PDF电子书基地 http://dayo1982.400gb.com

```
        byte[] b = new byte[50];
        dis.read(b , 2 , 10);      //将数据读到字节数组 b 中
        for (int i = 0; i < b.length; i++) {
            System.out.println(b[i]);
        }
        dis.close();
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.5.2 readByte 方法——读取此操作需要的字节

从所包含的输入流中读取此操作需要的字节。

方法声明

```
public final byte readByte() throws IOException
```

返回值：此输入流的下一个字节。

抛出异常：IOException。

示例

本示例通过 DataInputStream 类的 readByte()方法将指定磁盘文件中第一个字符输出。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println((char) dis.readByte());
```

3.5.3 readBoolean 方法——读取一个布尔值

从输出流中读取一个布尔值。

方法声明

```
public final boolean readBoolean() throws IOException
```

返回：读取的 boolean 值。

抛出异常：IOException。

示例

本示例通过 DataInputStream 类的 readBoolean()方法，读取磁盘文件中布尔值。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readBoolean());
```

3.5.4 readchar 方法——读取此操作需要的字符

从包含的输入流中读取此操作需要的字符。

方法声明

```
public final byte readchar() throws IOException
```

返回值：此输入流的下两个字节，即一个字符。

抛出异常：IOException。

示例

本示例通过 DataInputStream 类的 readchar()方法将指定磁盘文件中第一个字符输出。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readchar());
```

3.5.5 readDouble 方法——读取一个 double 值

从所包含的输入流中读取一个 double 值。

方法声明

```
public final double readDouble() throws IOException
```

返回值：此输入流的下 8 个字节，将它们解释为一个 double。

抛出异常

EOFException：如果此输入流已经到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生另其他 I/O 错误。

示例

本示例通过 DataInputStream 类的 readByte()方法将指定磁盘文件中第一个字符输出。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readDouble ());
```

3.5.6 readFloat 方法——读取一个 float 值

从输出流中读取一个单精度浮点值。

方法声明

```
public final boolean readFloat() throws IOException
```

返回：此输入流的下 4 个字节，将其解释为一个 float。

抛出异常：IOException。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例通过 DataInputStream 类的 readFloat()方法，读取一个 float 数。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readBoolean());
```

3.5.7 readFully 方法——读取一些字节

从输入流中读取一些字节，并将它们存储在缓冲区数组 b 中。读取的字节数等于 b 的长度。

方法声明 1

```
public final void readFully(byte[] b) throws IOException
```

入口参数：b 存储读取数据的缓冲区。

抛出异常

EOFException：如果此输入流已经到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生另其他 I/O 错误。

示例

本示例创建了一个文件输入流一个数据输入流，通过 DataInputStream 类的 read()方法将指定磁盘文件读取到字节数组 b 中（读取的长度为 b 的长度）并输出。

```
File file = new File("H:\\Getif.java");     // 用文件路径新建一个文件对象
try {
    //创建文件字节流
    FileInputStream fis = new FileInputStream(file);
    //创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    byte[] b = new byte[20];
    dis.readFully (b);        //将数据读到字节数组 b 中
    for (int i = 0; i < b.length; i++) {
        System.out.println(b[i]);
    }
    dis.close();
        fis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

方法声明 2

从输入流中读取 len 个字节。

```
public static File createTempFile(String prefix, String suffix, File directory)
    throws IOException
```

入口参数

b：存储读取数据的缓冲区。

off：指定数据中的偏移量的 int 值。

len：指定读取的字节数的 int 值。

抛出异常

EOFException：如果此输入流已经到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生另其他 I/O 错误。

示例

本示例创建了一个文件输入流一个数据输入流，通过 DataInputStream 类的 read() 方法将指定磁盘文件读取到字节数组 b 中（偏移量为 3 读取长度为 10 个字节）并输出。

```
File file = new File("H:\\Getif.java"); // 用文件路径新建一个文件对象
try {
    // 创建文件字节流
    FileInputStream fis = new FileInputStream(file);
    // 创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    byte[] b = new byte[20];
    dis.readFully(b, 3, 10); // 将数据读到字节数组 b 中
    for (int i = 0; i < b.length; i++) {
        System.out.println(b[i]);
    }
    dis.close();
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.5.8 readInt 方法——读取一个 int 值

从数据输入流中读取 int 值。

方法声明

```
public final byte readInt() throws IOException
```

返回值：此输入流的下 4 个字节，将其解释为一个 int。

抛出异常：IOException。

示例

本示例通过 DataInputStream 类的 readInt()方法，读取磁盘文件中 int 整数。

PDF电子书基地 http://dayo1982.400gb.com

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readchar());
```

3.5.9 readLong 方法——读取一个 long 值

从数据输入流中读取一个长整型。

方法声明

```
public final byte readLong() throws IOException
```

返回：此输入流的下 8 个字节，将其解释为一个 long。

抛出异常：IOException。

示例

本示例通过 DataInputStream 类的 readLong()方法，读取磁盘文件中 long 整数。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readLong());
```

3.5.10 readShort 方法——读取一个 short 值

读取两个输入字节并返回一个 short 值。

方法声明

```
public final short readShort() throws IOException
```

返回值：此输入流的下两个字节，将它们解释为一个有符号 16 位数。

抛出异常

EOFException：如果此输入流在读取这两个字节之前到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

示例

本示例通过 DataInputStream 类的 readShort()方法，读取磁盘文件中 short 数。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readShort ());
```

3.5.11 readUnsignedByte 方法——读取无符号的 byte 值

读取一个输入字节，将它左侧补零(zero-extend)转变为 int 类型，并返回结果，所以结果的范围是 0～255。如果接口 DataOutput 的 writeByte 方法的参数是 0～255 之间的值，则此方法适用于读取用 writeByte 写入的字节。

方法声明

```
public final int readUnsignedByte() throws IOException
```

返回值：此输入流的下一个字节，将它解释为一个无符号 8 位数。

抛出异常

EOFException：如果此输入流在读取这两个字节之前到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

示例

本示例通过 DataInputStream 类的 readUnsignedByte()方法，读取磁盘文件中字节，将它左侧补零（zero-extend）转变为 int 类型，并输出结果。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readUnsignedByte ());
```

3.5.12 readUnsignedShort 方法——读取无符号的 short 值

读取两个输入字节，并返回 0～65535 范围内的一个 int 值。设 a 为第一个读取字节，b 为第二个读取字节。返回的值是：(((a&0xff)<<8)|(b&0xff))。如果接口 DataOutput 的 writeShort 方法的参数是 0～65535 范围内的值，则此方法适用于读取用 writeShort 写入的字节。

方法声明

```
public final int readUnsignedShort () throws IOException
```

返回值：此输入流的下两个字节，将它们解释为一个无符号 16 位整数。

抛出异常：

EOFException：如果此输入流在读取这两个字节之前到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

示例

本示例通过 DataInputStream 类的 readUnsignedShort()方法，读取磁盘文件中字节，并输出结果。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readUnsignedShort ());
```

3.5.13 readUTF 方法——读取 UTF-8 编码的字符串

读入一个已使用 UTF-8 修改版格式编码的字符串。readUTF 的常规协定是：

PDF电子书基地 http://dayo1982.400gb.com

该方法读取使用 UTF-8 修改版格式编码的 Unicode 字符串的表示形式，然后以 String 的形式返回此字符串。

方法声明 1

```
public final String readUTF() throws IOException
```

返回值：一个 Unicode 字符串。

抛出异常

EOFException：如果此输入流在读取所有字节之前到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

TFDataFormatException：如果这些字节不表示一个有效的、UTF-8 修改版编码的字符串。

示例

本示例通过 DataOutputStream 向文件写入 Unicode 字符格式的数据，然后通过 DataInputStream 类的 readUTF()方法将指定磁盘文件的 Unicode 字符格式的数据并输出。

```
File file = new File("H:\\Getif.java"); // 用文件路径新建一个文件对象
try {
    // 创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    // 创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeUTF("明日科技");  //写入 Unicode 字符格式的数据
    // 创建文件输入流
    FileInputStream fis = new FileInputStream(file);
    // 创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    //读取 Unicode 字符格式的数据
    System.out.println(dis.readUTF());
    dos.close();
    dis.close();
    fos.close();
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

从流 in 中读取用 UTF-8 修改版格式编码的 Unicode 字符格式的字符串，然后以 String 形式返回此字符串。UTF-8 修改版表示形式的一些细节与 DataInput 的 readUTF 方法完全相同。

public static final String readUTF(DataInput in) throws IOException

入口参数：in 数据输入流。

返回值：一个 Unicode 字符串。

抛出异常

EOFException：如果此输入流在读取所有字节之前到达末尾。

IOException：该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

TFDataFormatException：如果这些字节不表示一个有效的、UTF-8 修改版编码的字符串。

示例

本示例通过 DataOutputStream 向文件写入 Unicode 字符格式的数据，然后通过 DataInputStream 类的 readUTF()方法将指定磁盘文件的 Unicode 字符格式的数据并输出。

```
File file = new File("H:\\Getif.java"); // 用文件路径新建一个文件对象
try {
    // 创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    // 创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeUTF("明日科技"); //写入 Unicode 字符格式的数据
    // 创建文件输入流
    FileInputStream fis = new FileInputStream(file);
    // 创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    //读取 Unicode 字符格式的数据
    String s =DataInputStream.readUTF(dis);
    System.out.println(s); //输出结果
    dos.close();
    dis.close();
    fos.close();
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.5.14 skipBytes 方法——在输入流中跳过数据的 n 个字节

试图在输入流中跳过数据的 n 个字节，并丢弃跳过的字节。不过，可以跳过更少的字节数，该字节数甚至可以为 0。这可能由很多情况引起；在已经跳过 n 个字节前到达文件末尾只是其中的一种可能。此方法从不抛出 EOFException。返回实际跳过的字节数。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public final int skipBytes(int n) throws IOException
```

入口参数：*n* 为要跳过的字节数。

返回值：实际跳过的字节数。

抛出异常：IOException，如果包含的输入流不支持查找操作；该流已关闭并且包含的输入流在关闭后不支持读取操作；发生其他 I/O 错误。

示例

本示例通过 DataInputStream 类的 skipBytes()方法跳过 10 个字节，然后开始读取文件，并输出了内容。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    //创建文件字节流
    FileInputStream fis = new FileInputStream(file);
    //创建数据输入流
    DataInputStream dis = new DataInputStream(fis);
    byte[] b = new byte[50];
    dis.skipBytes(10);      //跳过 10 个字节
    dis.read(b);      //将数据读到字节数组 b 中
    for (int i = 0; i < b.length; i++) {
        System.out.println(b[i]);
    }
    dis.close();
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6 DataOutputStream 类——数据输出流

java.io 包中的 DataOutputStream 类为数据输出流，该类允许应用程序以适当的方式将基本 Java 数据类型写入输出流中。

该类构造方法语法如下：

```
DataOutputStream(OutputStream out)
```

基于一个基础输出流，创建数据输出流实例。

3.6.1 flush 方法——清空此数据输出流

清空此数据输出流。这迫使所有缓冲的输出字节被写出到流中。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 write()方法，向指定磁盘文件中写数据，然后调用 flush()方法清空此数据输出流。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    byte[] bs = "本实例使用 DataOutputStream 输出流，在控制台输出字符串\n".getBytes();
    dos.write(bs);
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.2 size 方法——返回计数器 written 的当前值

返回计数器 written 的当前值，即到目前为止写入此数据输出流的字节数。如果计数器溢出，则将它包装到 Integer.MAX_VALUE。

方法声明

```
public final int size()
```

返回值：written 字段的值。

示例

本示例通过 DataOutputStream 类的 write()方法，向指定磁盘文件中写数据，并输出了写入的字节数。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    byte[] bs = "本实例使用 DataOutputStream 输出流，在控制台输出字符串\n".getBytes();
    dos.write(bs);
    System.out.println("数组长度为： " + bs.length);
    System.out.println("已输出字节数： " + dos.size());
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.6.3 write 方法(流输出方法)

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入基础输出流。如果没有抛出异常，则计数器 written 增加 len。

方法声明 1

```
public void write(byte[] b, int off, int len) throws IOException
```

入口参数

b：数据。

off：数据中的起始偏移量。

len：要写入的字节数。

抛出异常：IOException 如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 write()方法，向指定磁盘文件中写数据（数组的起始偏移量为 6，写入字节数为数组长度减 6）。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    byte[] bs = "本实例使用 DataOutputStream 输出流，在控制台输出字符串\n".getBytes();
    dos.write(bs, 6, bs.length - 6);
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

将指定字节（参数 b 的 8 个低位）写入基础输出流。如果没有抛出异常，则计数器 written 增加 1。

```
public void write(int b) throws IOException
```

入口参数：b 要写入的 byte。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 write()方法，向指定磁盘文件中写数据（数据的内容为 A～Z 的字母）。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
```

```
//创建文件输出流
FileOutputStream fos = new FileOutputStream(file);
//创建数据输出流
DataOutputStream dos = new DataOutputStream(fos);
for (int i = 65; i < 90; i++) {
    dos.write(i);    //写入数据
}
dos.flush();
dos.close();
fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.4 writeBoolean 方法——将 boolean 值写入输出流

将一个 boolean 值以 1-byte 值形式写入基础输出流。值 true 以值(byte)1 的形式被写出；值 false 以值(byte)0 的形式被写出。如果没有抛出异常，则计数器 written 增加 1。

方法声明

```
public final void writeBoolean(boolean v) throws IOException
```

入口参数：v 为要写入的 boolean 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeBoolean()方法，向指定磁盘文件中写入了布尔类型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeBoolean(true); //写入布尔类型的值
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.5 writeByte 方法——将 byte 值写入输出流

将一个 byte 值以 1-byte 值形式写出到基础输出流中。

方法声明

```
public final void writeByte(int i) throws IOException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：b 是要写入的整型数值。

抛出异常：IOException。

示例

本示例通过 DataOutputStream 类的 writeByte()方法，向指定磁盘文件中写数据。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
DataOutputStream dip = new DataOutputStream(fin);
dip.writeByte(99);
dip.close();
```

3.6.6 writeBytes 方法——将字符串中每一个字符的低字节的内容写入输出流

将字符串中每一个字符的低字节的内容写入目标设备中。

方法声明

```
public final void writeBytes(String str) throws IOException
```

参数说明：str 是要写入的字符串。

抛出异常：IOException。

示例

本示例通过 DataOutputStream 类的 writeBytes()方法，向指定磁盘文件中写入字符串数据。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
DataOutputStream dip = new DataOutputStream(fin);
dip.writeBytes("Java 编程词典");
dip.close();
```

3.6.7 writeChar 方法——将 char 值写入输出流

将一个 char 值以 2-byte 值形式写入基础输出流中，先写入高字节。如果没有抛出异常，则计数器 written 增加 2。

方法声明

```
public final void writeChar(int v) throws IOException
```

入口参数：v 为要写入的 char 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeChar()方法，向指定磁盘文件中写入了字符型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    char v = 'a';
    dos.writeChar(v); //写入字符型的数据
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.8 writechars 方法——将字符串每个 char 值写入输出流

将字符串中的每一个字符的两个字节的内容都写入目标设备中。

方法声明

```
public final void writechars(String str) throws IOException
```

参数说明：str 是要写入的字符串。

抛出异常：IOException。

示例

本示例通过 DataOutputStream 类的 writechars()方法，向指定磁盘文件中写入字符串数据。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
DataOutputStream dip = new DataOutputStream(fin);
dip.writechars("Java 编程词典");
dip.close();
```

3.6.9 writeDouble 方法——将 double 值写入输出流

使用 Double 类中的 doubleToLongBits 方法将 double 参数转换为一个 long 值，然后将该 long 值以 8-byte 值形式写入基础输出流中，先写入高字节。如果没有抛出异常，则计数器 written 增加 8。

方法声明

```
public final void writeDouble(double v) throws IOException
```

入口参数：v 为要写入的 double 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeDouble()方法，向指定磁盘文件中写

PDF电子书基地 http://dayo1982.400gb.com

入了 double 型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    double v = 123.456;
    dos.writeDouble(v); //写入 double 型的数据
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.10 writeFloat 方法——将 float 值写入输出流

使用 Float 类中的 floatToIntBits 方法将 float 参数转换为一个 int 值，然后将该 int 值以 4-byte 值形式写入基础输出流中，先写入高字节。如果没有抛出异常，则计数器 written 增加 4。

方法声明

```
public final void writeFloat(float v) throws IOException
```

入口参数：v 为要写入的 float 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeFloat()方法，向指定磁盘文件中写入了字 float 型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    float v = 123.456;
    dos.writeFloat (v); //写入 float 型的数据
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.11 writeLong 方法——将 long 值写入输出流

将一个 long 值以 8-byte 值形式写入基础输出流中，先写入高字节。如果没有抛出异常，则计数器 written 增加 8。

方法声明

```
public final void writeLong(long v) throws IOException
```

入口参数：v 为要写入的 long 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeLong()方法，向指定磁盘文件中写入了字 long 型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
    FileOutputStream fos = new FileOutputStream(file);
    //创建数据输出流
    DataOutputStream dos = new DataOutputStream(fos);
    long v = 123456;
    dos.writeLong (v); //写入 long 型的数据
    dos.flush();
    dos.close();
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.6.12 writeShort 方法——将 short 值写入输出流

将一个 short 值以 2-byte 值形式写入基础输出流中，先写入高字节。如果没有抛出异常，则计数器 written 增加 2。

方法声明

```
public final void writeShort(int v) throws IOException
```

入口参数：v 为要写入的 short 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例通过 DataOutputStream 类的 writeShort()方法，向指定磁盘文件中写入了字 short 型的数据。

```
File file = new File("H:\\Getif2.java");    // 用文件路径新建一个文件对象
try {
    //创建文件输出流
```


PDF电子书基地 http://dayo1982.400gb.com

```
        FileOutputStream fos = new FileOutputStream(file);
        //创建数据输出流
        DataOutputStream dos = new DataOutputStream(fos);
        short v = 123;
        dos.writeShort (v); //写入 short 型的数据
        dos.flush();
        dos.close();
        fos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.6.13 writeUTF()方法——将 UTF-8 编码字符写入输出流

该方法以与机器无关方式使用 UTF-8 编码写入基础输出流。

方法声明

```
public final void writeUTF(String str) throws IOException
```

入口参数：str 是要写入的字符串。

示例

本示例通过 DataOutputStream 类的 writeUTF()方法，向本地磁盘文件中写入数据。

```
File file = new File("H://Getif.java");
FileOutputStream fos = new FileOutputStream(file);
DataOutputStream dos = new DataOutputStream(fos);
os.writeUTF("写入文件的数据");
```

3.7 File 类——文件类

java.io 包中的 File 类用于封装系统的文件和目录的相关信息，例如文件大小、修改时间、文件路径等。

3.7.1 构造方法——File 类的构造方法

File 类的构造方法如下表所示。

File 类的构造方法

构 造 方 法	使 用 说 明
File(File parent,String child)	根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例
File(String pathname)	通过给定路径名字符串创建一个新的 File 实例
File(String pathname,String String child)	根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例
File(URI uri)	通过给定的 file，URI 转换为一个抽象路径名来创建一个新的 File 实例

3.7.2 常量——File类的常量

File类的常量如下表所示。

File的常量

常　量	含　有
pathSeparator	与系统有关的路径分隔符，返回字符串
pathSeparatorChar	与系统有关的路径分隔符，返回char对象
sparator	与系统有关的默认名称分隔符，返回一个字符串
separatorChar	与系统有关的默认名称分隔符，返回char对象

3.7.3 canExecute方法——判读文件是否可执行

测试应用程序是否可以执行此抽象路径名表示的文件。

方法声明

```
public boolean canExecute()
```

返回值：如果此抽象路径名指定的文件存在且可被应用程序读取时，返回true，否则返回false。

示例

本示例实现创建文件对象，并通过 canExecute()方法判断指定的文件是否可以在应用程序中执行，并给出相应的提示信息。

```
File file = new File("H:\\Getif.java");      //用文件路径新建一个文件对象
if(file. canExecute ()){
    System.out.println("应用程序可以执行此抽象路径名表示的文件");
}
else{
    System.out.println("应用程序不可以执行此抽象路径名下表示的文件");
}
```

3.7.4 canRead方法——判断文件是否可读

测试应用程序是否可以读取此抽象路径名表示的文件。

方法声明

```
public boolean canRead()
```

返回值：如果此抽象路径名指定的文件存在且可被应用程序读取时，返回true，否则返回false。

示例

本示例实现创建文件对象，并通过 canRead()方法判断指定的文件是否可以在应用程序中读取，并给出相应的提示信息。

PDF电子书基地 http://dayo1982.400gb.com

```
File file = new File("H://Getif.java");
if(file.canRead()){
    System.out.println("应用程序可以读取此抽象路径名表示的文件");
}
else{
    System.out.println("应用程序不可以读取此抽象路径名下表示的文件");
}
```

3.7.5 canWrite 方法——判断文件是否可写

测试应用程序是否可以修改指定抽象路径名表示的文件。

方法声明

```
public boolean canWrite()
```

返回值：如果相应的文件在文件系统中存在，并且允许应用程序对该文件进行写入时，返回 true，否则返回 false。

示例

本示例创建文件类对象，如果该文件存在，并且该文件允许在应用程序中进行写入，给出相应的输出信息。

```
File file = new File("H://Getif.java");
if(file.canWrite()){
    System.out.println("应用程序可以向指定的文件中写入信息");
}
else{
    System.out.println("应用程序不可以向指定的文件中写入信息");
}
```

3.7.6 compareTo 方法——比较文件对象

按字母顺序比较两个抽象路径名。此方法定义的顺序取决于底层系统。在 UNIX 系统上，比较路径名时，字母大小写通常很重要，而在 Windows 系统上，这通常不重要。

方法声明

```
public int compareTo(File pathname)
```

入口参数：pathname 为将与此抽象路径名进行比较的抽象路径名。

返回值：如果该参数等于此抽象路径名，则返回零；如果此抽象路径名在字母顺序上小于该参数，则返回小于零的值；如果此抽象路径名在字母顺序上大于该参数，则返回大于零的值。

示例

本示例实现创建两个文件对象，并通过 compareTo()方法进行比较，将返回值赋给 i。

```
String dir1 = "C:\\Getif.java";
String dir2 = "H:\\Getif.java";
File file1 = new File(dir1); //用文件路径新建一个文件对象
File file2 = new File(dir2); //用文件路径新建一个文件对象
int i = file1.compareTo(file2);//进行比较
```

3.7.7 createNewFile 方法——创建新的文件

当且仅当不存在具有此抽象路径名指定名称的文件时，创建该文件。

方法声明

```
public boolean createNewFile() throws IOException
```

返回值：如果指定的文件不存在并成功地创建，则返回 true；如果指定的文件已经存在，则返回 false。

抛出异常：IOException。

示例

本示例创建文件类对象，如果该文件对象不存在，在指定的路径下创建该文件。

```
File file = new File("E://myFirst.java");
if(!file.exists()){
    try {
        file.createNewFile();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.7.8 createTempFile 方法——创建临时文件

在默认临时文件目录中创建一个空文件，使用给定前缀和后缀生成其名称。调用此方法等同于调用 createTempFile(prefix,suffix,null)。

方法声明 1

```
public static File createTempFile(String prefix, String suffix)
    throws IOException
```

入口参数

prefix：用于生成文件名的前缀字符串；必须至少是三字符长。

suffix：用于生成文件名的后缀字符串；可以为 null，在这种情况下，将使用后缀".tmp"。

返回值：表示新建空文件的抽象路径名。

抛出异常

IllegalArgumentException：如果 prefix 参数包含的字符少于三个。

IOException：如果无法创建文件。

PDF电子书基地 http://dayo1982.400gb.com

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法不允许创建文件。

示例

本示例实现通过 createTempFile()方法在系统的临时目录创建了一个临时文件，扩展名为.txt。

```
try {
    File file = File.createTempFile("abc", ".txt");//创建临时文件
    System.out.println(file.exists());      //输出文件是否存在
} catch (IOException e) {
    e.printStackTrace();
}
```

示例

本示例实现通过 createTempFile()方法在系统的临时目录创建了一个临时文件，扩展名设置为 null,在这种情况下，将使用后缀".tmp"。

```
try {
    File file = File.createTempFile("abc", null);          //创建临时文件
    System.out.println(file.exists());      //输出文件是否存在
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

在指定目录中创建一个新的空文件，使用给定的前缀和后缀字符串生成其名称。

```
public static File createTempFile(String prefix, String suffix, File directory)
    throws IOException
```

入口参数

prefix：用于生成文件名的前缀字符串，必须至少是三字符长。

suffix：用于生成文件名的后缀字符串，可以为 null，在这种情况下，将使用后缀".tmp"。

directory：将创建的文件所在的目录；如果使用默认临时文件目录，则该参数为 null。

返回值：表示新建空文件的抽象路径名。

抛出异常

IllegalArgumentException：如果 prefix 参数包含的字符少于三个。

IOException：如果无法创建文件。

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法不允许创建文件。

说明

如果此方法成功返回，则可以保证：由返回的抽象路径名表示的文件在此方法被调用之前不存在。

此方法及其所有变体都不会在虚拟机的当前调用中再次返回相同的抽象路径名。

此方法只提供了临时文件的部分功能。要安排自动删除此方法创建的文件，可使用 deleteOnExit()方法。

prefix 参数至少必须是三个字节长。建议前缀使用一个短的、有意义的字符串，比如“hjb”或“mail”。suffix 参数可以为 null，在这种情况下，将使用后缀“.tmp”。要创建新文件，可能首先要调整前缀和后缀，使其满足底层平台的限制。如果前缀太长，则将它截断，但前三个字符将始终保留。如果后缀太长，则将它截断，但如果它以句点字符('.')开始，则该句点以及后跟的前三个字符将始终保留。进行了这些调整后，通过连接前缀、五个或更多个内部生成的字符以及后缀，便生成了新文件的名称。

如果 directory 参数为 null，则使用与系统有关的默认临时文件目录。默认临时文件目录由系统属性 java.io.tmpdir 指定。在 UNIX 系统上，此属性的默认值通常是“/tmp”或“/var/tmp”；在 Windows 系统上，该值通常是“C:\\WINNT\\TEMP”。在调用 Java 虚拟机时，可为此系统属性提供不同的值，但不保证使用程序更改此属性会对此方法使用的临时目录产生影响。

示例

本示例实现通过 createTempFile()方法在 C 盘创建了一个临时文件，扩展名为.txt。

```
try {
    //创建临时文件
    File file = File.createTempFile("abc", ".tmp",new File("c:\\"));
    System.out.println(file.exists());      //输出文件是否存在
} catch (IOException e) {
    e.printStackTrace();
}
```

示例

本示例实现通过 createTempFile()方法创建了一个临时文件，目录设置为 null，在这种情况下，将创建在在系统的临时目录。

```
try {
    File file = File.createTempFile("abc",".tmp", null);          //创建临时文件
    System.out.println(file.exists());      //输出文件是否存在
} catch (IOException e) {
```


PDF电子书基地 http://dayo1982.400gb.com

```
        e.printStackTrace();
    }
```

3.7.9 delete 方法——删除文件

删除此抽象路径名表示的文件或目录。如果此路径名表示一个目录，则该目录必须为空才能删除。

方法声明

```
public boolean delete()
```

返回值：当且仅当成功删除文件或目录时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法不允许创建文件。

示例

本示例实现通过首先创建了一个临时文件，然后调用 delete()方法删除文件并输出成功信息。

```
try {
    File file = File.createTempFile("abc", null); //创建临时文件
    if(file.delete())          //删除文件
        System.out.println("文件删除成功！ ");
} catch (IOException e) {
    e.printStackTrace();
}
```

3.7.10 deleteOnExit 方法——在虚拟机终止时，请求删除文件

在虚拟机终止时，请求删除此抽象路径名表示的文件或目录。文件（或目录）将以与注册相反的顺序删除。调用此方法删除已注册为删除的文件或目录无效。根据 Java 语言规范中的定义，只有在虚拟机正常终止时，才会尝试执行删除操作。一旦请求了删除操作，就无法取消该请求。所以应小心使用此方法。

方法声明

```
public void deleteOnExit()
```

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法不允许创建文件。

注意

此方法不应该用于文件锁定，因为所得协议可能无法可靠地工作。应该使用 FileLock 设施替代。

示例

本示例实现首先创建了一个临时文件，然后调用 deleteOnExit()方法在虚拟机终止时，删除文件。

```
try {
    File file = File.createTempFile("abc", null); //创建临时文件
    file.deleteOnExit();    //在虚拟机终止时，删除文件
} catch (IOException e) {
    e.printStackTrace();
}
```

3.7.11 equals 方法——判断 File 对象是否相等

测试此抽象路径名与给定对象是否相等。当且仅当该参数不是 null，而是一个与此抽象路径名表示相同的文件或目录的抽象路径名时，返回 true。两个抽象路径名是否相等取决于底层系统。在 UNIX 系统上，比较路径名时，字母大小写通常很重要，而在 Windows 系统上，这通常不重要。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要与此抽象路径名进行比较的对象。

返回值：如果对象相同，则返回 true，否则返回 false。

示例

本示例创建两个 File 对象，通过 equals 进行比较抽象路径名是否相等，将比较的结果赋给 b 并输出。

```
File file1 = new File("c:\\1.txt");   //用文件路径新建一个文件对象
File file2 = new File("c:\\1.txt");   //用文件路径新建一个文件对象
boolean b = file1.equals(file2);      //判断对象是否相等
```

3.7.12 exists 方法——判断文件是否存在

测试此抽象路径名表示的名称或目录是否存在。

方法声明

```
public boolean exists()
```

返回值：如果该抽象路径名表示的文件或目录存在时，返回 true，否则返回 false。

示例

本示例创建文件类对象。通过 exists()方法判断该文件是否存在，给出相应的提示信息。

```
File file = new File("E://myFirst.java");
if(file.exists()){
```


PDF电子书基地 http://dayo1982.400gb.com

```
        System.out.println("该文件存在！");
    }
    if(!file.exists()){
        System.out.println("该文件不存在！");
    }
```

3.7.13 getAbsoluteFile 方法——获取绝对路径的文件对象

返回此抽象路径名的绝对路径名形式。

方法声明

```
public File getAbsoluteFile()
```

返回值：绝对抽象路径名，它与此抽象路径名表示相同的文件或目录。

抛出异常为：SecurityException，如果无法访问所需的系统属性值。

示例

本示例创建两个 File 对象，file2 是通过 File 的 getAbsoluteFile()方法返回的。

```
File file1 = new File("Test.java"); //用文件相对路径新建一个文件对象
File file2 = file1.getAbsoluteFile();
```

3.7.14 getAbsolutePath 方法——获取绝对路径的字符串对象

返回此抽象路径名的绝对路径名字符串。

方法声明

```
public String getAbsolutePath()
```

返回值：绝对路径名字符串，它与此抽象路径名表示相同的文件或目录。

抛出异常：SecurityException，如果无法访问所需的系统属性值。

说明

如果此抽象路径名已经是绝对路径名，则返回该路径名字符串，这与 getPath()方法一样。如果此抽象路径名是空抽象路径名，则返回当前用户目录的路径名字符串，该目录由系统属性 user.dir 指定。否则，使用与系统有关的方式解析此路径名。在 UNIX 系统上，根据当前用户目录解析相对路径名，可使该路径名成为绝对路径名。在 Windows 系统上，根据路径名指定的当前驱动器目录（如果有）解析相对路径名，可使该路径名成为绝对路径名；否则，可以根据当前用户目录解析它。

示例

本示例创建一个 File 对象，通过 File 的 getAbsolutePath()方法获取了文件的绝对路径。

```
File file1 = new File("Test.java"); //用文件路径新建一个文件对象
String path = file1.getAbsolutePath(); //获取文件的绝对路径
```

3.7.15 getCanonicalFile 方法——返回此抽象路径名的规范形式

返回此抽象路径名的规范形式。

方法声明

```
public File getCanonicalFile() throws IOException
```

返回值：规范路径名字符串，它与此抽象路径名表示相同的文件或目录。

抛出异常：

IOException：如果发生 I/O 错误（可能是因为构造规范路径名需要进行文件系统查询）。

SecurityException：如果无法访问所需的系统属性值，或者存在安全管理器，且其 SecurityManager.checkRead(java.io.FileDescriptor)方法拒绝对文件进行读访问。

说明

如果此抽象路径名已经是绝对路径名，则返回该路径名字符串，这与 getPath()方法一样。如果此抽象路径名是空抽象路径名，则返回当前用户目录的路径名字符串，该目录由系统属性 user.dir 指定。否则，使用与系统有关的方式解析此路径名。在 UNIX 系统上，根据当前用户目录解析相对路径名，可使该路径名成为绝对路径名。在 Windows 系统上，根据路径名指定的当前驱动器目录（如果有）解析相对路径名，可使该路径名成为绝对路径名；否则，可以根据当前用户目录解析它。

示例

本示例创建一个 File 对象，file2 是通过 File 的 getCanonicalFile()方法返回的。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
try {
    File file2 = file1.getCanonicalFile(); //获取抽象路径名的规范形式
} catch (IOException e) {
    e.printStackTrace();
}
```

3.7.16 getCanonicalPath 方法——返回此抽象路径名的规范路径名字符串

返回此抽象路径名的规范路径名字符串。

方法声明

```
public String getCanonicalPath() throws IOException
```

返回值：规范路径名字符串，它与此抽象路径名表示相同的文件或目录。

抛出异常

IOException：如果发生 I/O 错误（可能是因为构造规范路径名需要进行文件

PDF电子书基地 http://dayo1982.400gb.com

系统查询)。

SecurityException：如果无法访问所需的系统属性值，或者存在安全管理器，且其 SecurityManager.checkRead(java.io.FileDescriptor)方法拒绝对文件进行读访问。

说明

规范路径名是绝对路径名，并且是惟一的。规范路径名的准确定义与系统有关。如有必要，此方法首先将路径名转换为绝对路径名，这与调用 getAbsolutePath()方法的效果一样，然后用与系统相关的方式将它映射到其惟一路径名。这通常涉及从路径名中移除多余的名称（比如“.”和“..”）、解析符号连接（对于 UNIX 平台），以及将驱动器号转换为标准大小写形式（对于 Windows 平台）。每个表示现存文件或目录的路径名都有一个惟一的规范形式。每个表示不存在文件或目录的路径名也有一个惟一的规范形式。不存在文件或目录路径名的规范形式可能不同于创建文件或目录之后同一路径名的规范形式。同样，现存文件或目录路径名的规范形式可能不同于删除文件或目录之后同一路径名的规范形式。

示例

本示例创建一个 File 对象，通过 File 的 getCanonicalPath()方法返回抽象路径名的规范路径名字符串。

```
File file1 = new File("Test.java"); //用文件相对路径新建一个文件对象
try {
    String path = file1.getCanonicalPath();      //获取抽象路径名的规范路径名
    System.out.println(path);
} catch (IOException e) {
    e.printStackTrace();
}
```

3.7.17 getFreeSpace 方法——获取剩余空间

返回此抽象路径名指定的分区中未分配的字节数。

方法声明

```
public long getFreeSpace()
```

返回值：分区上的未分配字节数；如果抽象路径名没有指定分区，则返回 0L。返回值将小于等于 getTotalSpace()返回的总文件系统大小。

抛出异常

SecurityException：如果安装了安全管理器，并且安全管理器拒绝。RuntimePermission("getFileSystemAttributes")，或者其 SecurityManager.

checkRead(String)方法拒绝对此抽象路径名指定的文件进行读访问。

说明

返回的未分配字节数是一个提示，而不是一个保证，只能使用这些字节的一部分。未分配字节数很可能在此调用后立即与实际相符。某些外部 I/O 操作可能使其与实际不符，这些操作包括那些在此虚拟机外部系统上进行的操作。此方法不保证对此文件系统的写操作将成功。

示例

本示例创建一个 File 对象，通过 File 的 getFreeSpace()方法获取 file1 所在分区未分配的字节数。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
long l = file1.getFreeSpace(); //获取 file1 所在分区未分配的字节数
```

3.7.18 getName 方法——获取文件名称

该方法获取文件的名称，并以字符串形式将文件名称返回。

方法声明

```
public String getName()
```

返回值：此抽象路径名表示的文件或目录的名称；如果路径名的名称序列为空，则返回空字符串。

示例

本示例创建文件类对象，并通过 getName()方法获取此文件对象的文件名称。

```
File file = new File("E://myFirst.java");
String strName = file.getName();
```

3.7.19 getParent 方法——获取上一级路径的字符串

返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null。抽象路径名的父路径名由路径名的前缀（如果有），以及路径名名称序列中最后一个名称以外的所有名称组成。如果名称序列为空，那么该路径名没有指定父目录。

方法声明

```
public String getParent()
```

返回值：此抽象路径名指定父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null。

示例

本示例创建一个 File 对象，通过 File 的 getParent()方法获取 file1 的父

PDF电子书基地 http://dayo1982.400gb.com

目录。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
System.out.println("父目录：" + file1.getParent());    //获取文件的父目录
```

3.7.20 getParentFile 方法——获取上一级路径的 File 对象

返回此抽象路径名父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null。抽象路径名的父路径名由路径名的前缀（如果有），以及路径名名称序列中最后一个名称以外的所有名称组成。如果名称序列为空，那么该路径名没有指定父目录。

方法声明

```
public File getParentFile()
```

返回值：此抽象路径名指定父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null。

示例

本示例创建一个 File 对象，通过 File 的 getParentFile()方法返回了 file1 父目录的 File 对象。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
File file2 = file1.getParentFile();  //返回 file1 父目录的 File 对象
```

3.7.21 getPath 方法——获取路径字符串

将此抽象路径名转换为一个路径名字符串，所得字符串使用默认名称分隔符分隔名称序列中的名称。

方法声明

```
public String getPath()
```

返回值：此抽象路径名的字符串形式。

示例

本示例创建一个 File 对象，通过 File 的 getPath()方法输出了 file1 的路径。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
System.out.println("文件路径：" + file1.getPath()); //获取文件路径
```

3.7.22 getTotalSpace 方法——获取总空间大小

返回此抽象路径名指定的分区大小。

方法声明

```
public long getTotalSpace()
```

返回值：分区的大小，以字节为单位；如果此抽象路径名没有指定分区，则返回 0L。

示例

本示例创建一个 File 对象，通过 File 的 getTotalSpace()方法输出了 file1 所在路径的分区大小。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
System.out.println(file1.getTotalSpace());//获取 file1 路径分区的大小
```

3.7.23 getUsableSpace 方法——精确获取可写入空间大小

返回此抽象路径名指定的分区上可用于此虚拟机的字节数。若有可能，此方法将检查写权限和其他操作系统限制，因此与 getFreeSpace()相比，此方法能更准确地估计可实际写入的新数据数。

方法声明

```
public long getUsableSpace()
```

返回值：分区上的可用字节数；如果抽象路径名没有指定分区，则返回 0L。在此信息不可用的系统上，此方法等效于调用 getFreeSpace()。

抛出异常

SecurityException：如果安装了安全管理器，并且安全管理器拒绝 RuntimePermission("getFileSystemAttributes")或者其 SecurityManager.checkRead(String)方法拒绝对此抽象路径名指定的文件进行读访问。

注意

返回的可用字节数是一个提示，而不是一个保证，只能使用这些字节的一部分。未分配字节数很可能在此调用后立即与实际相符。某些外部 I/O 操作可能使其与实际不符，这些操作包括那些在此虚拟机外部系统上进行的操作。此方法不保证对此文件系统的写操作将成功。

示例

本示例创建一个 File 对象，通过 File 的 getUsableSpace()方法输出了 file1 所在分区可用于此虚拟机的字节数。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
//获取 file1 分区上可用于此虚拟机的字节数
System.out.println(file1.getUsableSpace());
```

3.7.24 hashCode 方法——生成 File 对象的哈希码

计算此抽象路径名的哈希码。因为抽象路径名的相等性与系统有关，所以对其哈希码的计算也与系统有关。在 UNIX 系统上，抽象路径名的哈希码等于其路径名字符串和十进制值 1234321 的哈希码的异或。在 Windows 系统上，哈希码等于其转换为小写的路径名字符串和十进制值 1234321 的哈希码的异或。在将路

PDF电子书基地 http://dayo1982.400gb.com

径名字符串转换为小写时不考虑语言环境。

方法声明

```
public int hashCode()
```

返回值：此抽象路径名的哈希码。

示例

本示例创建一个 File 对象，通过 File 的 hashCode()方法输出了 file1 路径名的哈希码。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
System.out.println(file1.hashCode()); //输出哈希码
```

3.7.25 isAbsolute 方法——测试此路径名是否为绝对路径名

测试此抽象路径名是否为绝对路径名。绝对路径名的定义与系统有关。在 UNIX 系统上，如果路径名的前缀是“/”，那么该路径名是绝对路径名。在 Microsoft Windows 系统上，如果路径名的前缀是后跟“\\”的盘符，或者是“\\\\”，那么该路径名是绝对路径名。

方法声明

```
public boolean isAbsolute()
```

返回值：如果此抽象路径名是绝对路径名，则返回 true；否则返回 false。

示例

本示例创建一个 File 对象，通过 File 的 isAbsolute()方法输出了 file1 路径是否为绝对路径。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
if(file1.isAbsolute())//文件路径是否是绝对路径
    System.out.println("是绝对路径");
else
    System.out.println("不是绝对路径");
```

示例

本示例创建一个 File 对象，通过 File 的 isAbsolute()方法输出了 file1 路径是否为绝对路径。

```
File file1 = new File("\\");  //用文件路径新建一个文件对象
if(file1.isAbsolute())//文件路径是否是绝对路径
    System.out.println("是绝对路径");
else
    System.out.println("不是绝对路径");
```

3.7.26 isDirectory 方法——判断 File 对象是否为目录

测试此抽象路径名表示的文件是否是一个目录。

方法声明

```
public boolean isDirectory()
```

返回值：如果指定抽象路径名表示的文件存在并且是一个目录时，返回 true，否则返回 false。

示例

本示例实现创建 File 实例，并通过 isDirectory()方法判断 File 实例是否是一个目录，并给出相应的信息。

```
File file = new File("H://word");
if(file.isDirectory()){
    System.out.println("file 是一个目录");
}
```

3.7.27 isFile 方法——判断 File 对象是否为文件

测试此抽象路径名表示的文件是否是一个标准文件。如果该文件不是一个目录，并且满足其他与系统有关的标准，那么该文件是标准文件。由 Java 应用程序创建的所有非目录文件一定是标准文件。

方法声明

```
public boolean isFile()
```

返回值：当且仅当此抽象路径名表示的文件存在且是一个标准文件时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 isFile()方法输出了 file1 路径是否为一个标准文件。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
if(file1.isFile())       //file1 是否为一个标准文件
    System.out.println("是一个标准文件");
else
    System.out.println("不是一个标准文件");
```

3.7.28 isHidden 方法——判断 File 对象是否设置隐藏属性

测试此抽象路径名指定的文件是否是一个隐藏文件。隐藏的具体定义与系统有关。在 UNIX 系统上，如果文件名以句点字符('.')开头，则认为该文件被隐藏。在 Windows 系统上，如果在文件系统中文件被标记为隐藏，则认为该文件被隐藏。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public boolean isHidden()
```

返回值：当且仅当此抽象路径名表示的文件根据底层平台约定是隐藏文件时，返回 true。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 isHidden()方法输出了 file1 路径是否为一个隐藏文件。

```
File file1 = new File("C:\\Getif.java");    //用文件路径新建一个文件对象
if(file1.isHidden())  //file1 是否为一个隐藏文件
    System.out.println("是一个隐藏文件");
else
    System.out.println("不是一个隐藏文件");
```

3.7.29 lastModified 方法——获取最后修改时间

获取此抽象路径名表示的文件的最后一个修改时间。

方法声明

```
public long lastModified()
```

返回值：文件最后一次被修改的时间的 long 值，与时间点（1970 年 1 月 1 日 00：00：00GMT）之间的毫秒数。如果指定的文件不存在，或发生 I/O 异常，则返回 0L。

示例

本示例创建 File 对象，并通过 lastModified()方法获取文件最后一次的修改时间。

```
File file = new File("H://word");
long len = file.lastModified();
String modifDate = new Date(file.lastModified())
.toLocaleString();
```

3.7.30 length 方法——获取文件长度

获取指定文件抽象路径名表示的文件的长度。但此方法不能确定抽象路径名表示的目录的长度。

方法声明

```
public long length()
```

返回值：以字节为单位获取抽象路径名表示的文件的长度；如果文件不存在，则返回 0L。

示例

本示例创建文件类对象，并通过 length()方法获取此文件的长度，并在控制台上输出提示信息。

```
File file = new File("H://Getif.java");
long length = file.length();
System.out.println("Getif.java 文件的长度是"+length+"B");
```

3.7.31 list 方法——获取目录列表的字符串数组

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。

方法声明 1

```
public String[] list()
```

返回值：字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果此抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

注意

如果此抽象路径名不表示一个目录，那么此方法将返回 null；否则返回一个字符串数组，每个数组元素对应目录中的每个文件或目录。表示目录本身及其父目录的名称不包括在结果中。每个字符串是一个文件名，而不是一条完整路径。不保证所得数组中的相同字符串将以特定顺序出现，特别是不保证它们按字母顺序出现。

示例

本示例创建一个 File 对象，通过 File 的 list()方法输出 C 盘的所有文件和目录的名字。

```
String[] s;
File file1 = new File("C:\\ ");        //用文件路径新建一个文件对象
s = file1.list();          //获取目录中的文件和目录
if(s != null){
      for (int i = 0; i < s.length; i++) {
            System.out.println(s[i]); //输出获取到的文件和目录
      }
}
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录。

```
public String[] list(FilenameFilter filter)
```

入口参数：filtero 为文件名过滤器。

返回值：字符串数组，这些字符串指定此抽象路径名表示的目录中给定 filter 能接受的文件和目录。如果目录为空，或者没有名称被过滤器接受，那么该数组将为空。如果抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

注意

除了返回数组中的字符串必须满足过滤器外，此方法的行为与 list()方法相同。如果给定 filter 为 null，则接受所有名称；否则，当且仅当在此抽象路径名及其表示的目录中的文件名或目录名上调用过滤器的 FilenameFilter.accept(java.io.File,java.lang.String)方法返回 true 时，该名称才满足过滤器。

示例

本示例创建过滤器，通过 File 的 list()方法输出 C 盘的所有以“.txt”结尾的文件和目录的名字。

```
// 内部类实现了 FilenameFilter 接口用于过滤文件
static class MyFilter implements FilenameFilter {
    public boolean accept(File dir, String name) {
return name.endsWith(".txt");
    }
}
public static void main(String[] args) {
    String[] s;
    File file1 = new File("C:\\"); // 用文件路径新建一个文件对象
    // 获取目录中经过过滤后的文件和目录
    s = file1.list(new MyFilter());
    if (s != null) {
        for (int i = 0; i < s.length; i++) {
            System.out.println(s[i]); // 输出获取到的文件
        }
    }
}
```

3.7.32 listFiles 方法——获取目录列表的 File 数组

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定

过滤器的文件和目录。除了返回数组中的路径名必须满足过滤器外，此方法的行为与 listFiles()方法相同。如果给定 filter 为 null，则接受所有路径名；否则，当且仅当在路径名上调用过滤器的 FileFilter.accept(java.io.File)方法返回 true 时，该路径名才满足过滤器。

方法声明 1

```
public File[] listFiles(FileFilter filter)
```

入口参数：filter 为文件过滤器。

返回值：抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个文件过滤器，通过 File 的 listFiles()方法输出 C 盘所有隐藏文件和目录的路径。

```
// 内部类实现了 FileFilter 接口用于过滤文件
static class MyFilter implements FileFilter {
    public boolean accept(File pathname) {
        return pathname.isHidden();
    }
}
public static void main(String[] args) {
    File[] fs;
    File file1 = new File("C:\\"); // 用文件路径新建一个文件对象
    // 获取目录中经过过滤后的文件
    fs = file1.listFiles(new MyFilter());
    if (fs != null) {
        for (int i = 0; i < fs.length; i++) {
            System.out.println(fs[i]); // 输出获取到的文件和目录
        }
    }
}
```

方法声明 2

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。除了返回数组中的路径名必须满足过滤器外，此方法的行为与 listFiles()方法相同。如果给定 filter 为 null，则接受所有路径名；否则，当且仅当在此抽象路径名及其表示的目录中的文件名或目录名上调用过滤器的

PDF电子书基地 http://dayo1982.400gb.com

FilenameFilter.accept(java.io.File,java.lang.String)方法返回 true 时，该路径名才满足过滤器。

```
public File[] listFiles(FilenameFilter filter)
```

入口参数：filter 为文件名过滤器。

返回值：抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果此抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建过滤器，通过 File 的 listFiles()方法输出 C 盘的所有以“.txt”结尾的文件和目录的路径。

```
// 内部类实现了 FilenameFilter 接口用于过滤文件
static class MyFilter implements FilenameFilter {
    public boolean accept(File dir, String name) {
        return name.endsWith(".txt");
    }
}
public static void main(String[] args) {
    File[] fs;
    File file1 = new File("C:\\"); // 用文件路径新建一个文件对象
    // 获取目录中经过过滤后的文件
    fs = file1.listFiles(new MyFilter());
    if (fs != null) {
        for (int i = 0; i < fs.length; i++) {
            System.out.println(fs[i]); // 输出获取到的文件和目录
        }
    }
}
```

3.7.33 listRoots 方法——获取根目录的 File 数组

列出可用的文件系统根。

方法声明

```
public static File[] listRoots()
```

返回值：表示可用文件系统根的 File 对象数组；如果无法确定根集，则返回 null。如果没有文件系统，那么该数组将为空。

说明

特定 Java 平台可以支持零个或更多个分层组织的文件系统。每个文件系统

有一个 root 目录，可以从这里到达文件系统中的所有其他文件。例如，Windows 平台为每个活动驱动器提供了一个根目录；UNIX 平台只有一个根目录，即“/”。可用文件系统根的设置受各种系统级操作的影响，比如可移动介质的插入和弹出，以及断开或卸载那些物理磁盘或虚拟磁盘。

此方法返回一个 File 对象数组，这些对象表示可用文件系统根的根目录。可以保证本地机器上物理存在的任何文件的规范路径名都以此方法返回的根之一开始。

位于其他一些机器上的文件的规范路径名是通过远程文件系统协议（比如 SMB 或 NFS）访问的，它们可能以此方法返回的根之一开始，也可能不是这样。如果远程文件的路径名在语法上无法与本地文件的路径名进行区分，那么它将以此方法返回的根之一开始。例如，此方法将返回表示 Windows 平台上映射为网络驱动器根目录的 File 对象，而不返回包含 UNC 路径名的 File 对象。与此类中的大多数方法不同，此方法不抛出安全性异常。如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String)方法拒绝对特定根目录进行读访问，那么该目录将不会出现在结果中。

示例

本示例通过 File 的 listRoots()方法获取可用的文件系统根并依次输出。

```
File[] fs;
fs = File.listRoots(); //获取可用的文件系统根
if (fs != null) {
    for (int i = 0; i < fs.length; i++) {
        System.out.println(fs[i]); // 列出可用的文件系统根
    }
}
```

3.7.34 mkdir 方法——创建目录

创建此抽象路径名指定的目录。

方法声明

```
public boolean mkdir()
```

返回值：当且仅当已创建目录时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 mkdir()方法创建了一个文件夹。

PDF电子书基地 http://dayo1982.400gb.com

```
File file = new File("C:\\明日科技");      // 用文件路径新建一个文件对象
if(file.mkdir()){
    System.out.println("目录创建成功");
}
```

3.7.35 mkdirs 方法——创建完整路径的所有目录

创建此抽象路径名指定的目录，包括所有必需但不存在的父目录。注意，此操作失败时也可能已经成功地创建了一部分必需的父目录。

方法声明

```
public boolean mkdirs()
```

返回值：当且仅当已创建目录以及所有必需的父目录时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 mkdirs()方法创建了指定的目录，包括所有必需但不存在的父目录。

```
File file = new File("C:\\明日科技\\java\\j2ee");
if(file.mkdirs()){
    System.out.println("目录创建成功");
}
```

3.7.36 renameTo 方法——重命名文件或文件夹

重新命名此抽象路径名表示的文件。此方法行为的许多方面都是与平台有关的：重命名操作无法将一个文件从一个文件系统移动到另一个文件系统，该操作不是不可分的，如果已经存在具有目标抽象路径名的文件，那么该操作可能无法获得成功。应该始终检查返回值，以确保重命名操作成功。

方法声明

```
public boolean renameTo(File dest)
```

入口参数：dest 为指定文件的新抽象路径名。

返回值：当且仅当重命名成功时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 renameTo()方法将 C 盘下的“明日

科技”文件夹重命名为“明日图书”。

```
File file = new File("C:\\明日科技");      // 用文件路径新建一个文件对象
if(file.renameTo(new File("C:\\明日图书"))){
    System.out.println("重命名成功！");
}
```

3.7.37 setExecutable 方法——设置可执行属性

设置此抽象路径名所有者执行权限的一个便捷方法。

方法声明 1

```
public boolean setExecutable(boolean executable)
```

入口参数：executable 如果为 true，则设置允许执行操作的访问权限；如果为 false，则不允许执行操作。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 executable 为 false，并且底层文件系统不实现执行权限，那么操作也将失败。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setExecutable()方法设置文件的执行权限。

```
File file = new File("H:\\Getif.java");    // 用文件路径新建一个文件对象
if(file.setExecutable(true)){
    System.out.println("允许执行权限");
}
```

方法声明 2

设置此抽象路径名的所有者或所有用户的执行权限。

```
public static File createTempFile(String prefix, String suffix, File directory)
```

入口参数

executable：如果为 true，则设置允许执行操作的访问权限；如果为 false，则不允许执行操作。

ownerOnly：如果为 true，则执行权限只适用于所有者的执行权限；否则适用于所有用户。如果底层文件系统不能区分所有者执行权限与其他执行权限，那么无论该参数为何值，执行权限将适用于所有用户。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 executable 为 false，并且底层文件系统不实现执行权限，那么操作也将失败。

PDF电子书基地 http://dayo1982.400gb.com

抛出异常：

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setExecutable()方法设置文件的所有者执行权限。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
if(file.setExecutable(true,true)){
    System.out.println("允许所有者执行权限");
}
```

3.7.38 setLastModified 方法——设置最后修改时间

设置此抽象路径名指定的文件或目录的最后一次修改时间。

方法声明

```
public boolean setExecutable(boolean executable)
```

入口参数：time 新的最后一次修改时间，用与时间点（1970 年 1 月 1 日，00:00:00GMT）之间的毫秒数表示。

返回值：当且仅当该操作成功时，返回 true；否则返回 false。

抛出异常

IllegalArgumentException：如果该参数为负。

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法拒绝对指定文件进行写访问。

示例

本示例创建一个 File 对象，通过 File 的 setLastModified()方法设置文件最后一次修改时间。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
//设置文件最后一次修改时间
if(file.setLastModified(new Date().getTime())){
    System.out.println("修改完成！");
}
```

3.7.39 setReadable 方法——设置可读属性

设置此抽象路径名所有者读权限的一个便捷方法。

方法声明 1

```
public boolean setReadable(boolean readable)
```

入口参数：readable 如果为 true，则设置允许读操作的访问权限；如果为

false，则不允许读操作。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 readable 为 false，并且底层文件系统不实现读权限，那么操作也将失败。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setReadable()方法设置文件的读权限。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
if(file. setReadable (true)){
      System.out.println("允许读权限");
}
```

方法声明 2

设置此抽象路径名的所有者或所有用户的读权限。

```
public boolean setReadable(boolean readable, boolean ownerOnly)
```

入口参数

readable 如果为 true，则设置允许读操作的访问权限；如果为 false，则不允许读操作。

ownerOnly 如果为 true，则读权限只适用于所有者的读权限；否则适用于所有用户。如果底层文件系统不能区分所有者读权限与其他读权限，那么无论该参数为何值，读权限将适用于所有用户。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 readable 为 false，并且底层文件系统不实现读权限，那么操作也将失败。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setReadable()方法设置文件的所有者读权限。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
if(file.setExecutable(true,true)){
      System.out.println("允许所有者读权限");
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.7.40 setReadOnly 方法——设置只读属性

标记此抽象路径名指定的文件或目录，从而只能对其进行读操作。调用此方法后，可以保证在被删除或被标记为允许写访问之前，文件或目录不会发生更改。是否可以删除某个只读文件或目录则取决于底层系统。

方法声明

```
public boolean setReadOnly()
```

返回值：当且仅当该操作成功时，返回 true；否则返回 false。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkWrite (java.lang.String)方法拒绝对指定文件进行写访问。

示例

本示例创建一个 File 对象，通过 File 的 setReadOnly()方法设置文件为只读属性。

```
File file = new File("H:\\Getif.java");
//设置文件为只读属性
if(file.setReadOnly()){
    System.out.println("设置文件为只读属性！");
}
```

3.7.41 setWritable 方法——设置可写属性

设置此抽象路径名所有者写权限的一个便捷方法。

方法声明 1

```
public boolean setWritable(boolean writable)
```

入口参数：writable 如果为 true，则设置允许写操作的访问权限；如果为 false，则不允许写操作。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setWritable()方法设置文件的写的权限。

```
File file = new File("H:\\Getif.java");     // 用文件路径新建一个文件对象
if(file.setWritable(true)){
```

```
        System.out.println("允许写的权限");
    }
```

方法声明 2

设置此抽象路径名的所有者或所有用户的写权限。

```
public boolean setWritable(boolean writable, boolean ownerOnly)
```

入口参数

writable：如果为 true，则设置允许写操作的访问权限；如果为 false，则不允许写操作。

ownerOnly：如果为 true，则写权限只适用于所有者的写权限；否则适用于所有用户。如果底层文件系统不能区分所有者写权限与其他写权限，那么无论该参数为何值，写权限将适用于所有用户。

返回值：当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。

抛出异常

SecurityException：如果存在安全管理器，且其 SecurityManager.checkRead (java.lang.String)方法拒绝对文件进行读访问。

示例

本示例创建一个 File 对象，通过 File 的 setWritable()方法设置文件的所有者写的权限。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
if(file.setWritable(true,true)){
    System.out.println("允许所有者写权限");
}
```

3.7.42 toString 方法——File 的字符串格式

返回此抽象路径名的路径名字符串。该字符串就是 getPath()方法返回的字符串。

方法声明

```
public String toString()
```

返回值：此抽象路径名的字符串形式。

示例

本示例创建一个 File 对象，通过 File 的 toString()方法返回 File 对象径名的路径名字符串。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
String s = file.toString();
```


PDF电子书基地 http://dayo1982.400gb.com

3.7.43 toURI 方法——获取 File 的 URI

构造一个表示此抽象路径名的 file:URI，该 URI 的具体形式与系统有关。如果可以确定此抽象路径名表示的文件是一个目录，那么所得 URI 将以斜杠结束。

方法声明

```
public URI toURI()
```

返回值：一个绝对分层 URI，由一个等于“file”的 scheme、表示此抽象路径名的 path，以及未定义的 authority、query 和 fragment 组件组成。

抛出异常：SecurityException 如果不能访问请求的系统属性值。

示例

本示例创建一个 File 对象，通过 File 的 toURI()方法返回了一个 URI。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
URI u = file.toURI();          //返回一个 URI
```

3.8 FileInputStream 类——文件字节输入流

程序在运行期间，大部分数据都是在内存中进行操作，当程序结束或关闭时，这些数据将完全消失，但是有些数据需要永久保存，例如程序的配置信息、日志信息等。通过使用文件输入输出流，可以和指定的文件建立关联，把需要永久保存的数据输出到文件中，程序在下次运行时，可以从文件中取回这些数据。

java.io 包中的 FileInputStream 类实现从文件系统中某个文件中获取输入字节。

3.8.1 构造方法——FileInputStream 类的构造方法

该类的构造方法如下表所示。

FileInputStream 类的构造方法

构 造 方 法	使 用 说 明
FileInputStream(File file)	创建一个使用默认大小输入缓冲区的缓冲字符输出流
FileInputStream(FileDescriptor fdObjp)	通过使用文件描述符 fdObjp 创建 FileInputStream
FileInputStream(String name)	通过打开一个到实际文件的连接来创建一个 FileInputStream。该文件通过文件系统中的路径名 name 指定

3.8.2 available 方法——获取可读的有效字节数

返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取（或跳过）的估计剩余字节数。下一次调用可能是同一个线程，也可能是另一个线程。一次读取或跳过此数量个字节不会发生阻塞，但读取或跳过的字节可能小于该

数。在某些情况下，非阻塞的读取（或跳过）操作在执行很慢时看起来受阻塞，例如，在网速缓慢的网络上读取大文件时。

方法声明

```
public int available() throws IOException
```

返回值：可以不受阻塞地从此输入流中读取（或跳过）的估计剩余字节数。

抛出异常：IOException 如果此文件输入流已通过调用 close 关闭，或者发生 I/O 错误。

示例

本示例创建文件字节输入流对象，并通过 available()方法估计了剩余的字节数，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    FileInputStream fi = new FileInputStream(file);
    System.out.println("文件大小为" + file.length() + "Bytes");
    fi.read();//读取一个字节
    System.out.println("估计文件还有" + fi.available() + "Bytes");
    fi.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.8.3 close 方法——关闭文件输入流

关闭此文件输入流并释放与此流有关的所有系统资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例创建文件字节输入流对象，并通过该对象读取指定文件中数据，最后调用 close()方法将流关闭。

```
File file = new File("H://Getif.java");
try {
    FileInputStream fis = new FileInputStream(file);
    char a;
    for (int i = 0; i < file.length(); i++) {
        a = (char) fis.read();
        System.out.print(a);
    }
    fis.close();
} catch (Exception e) {
```


PDF电子书基地 http://dayo1982.400gb.com

```
        e.printStackTrace();
    }
```

3.8.4 finalize 方法——在销毁改对象时关闭输入流

确保在不再引用文件输入流时调用其 close 方法。由于重写了 Object 的 finalize 方法。所以在垃圾回收时会自动调用。

方法声明

```
protected void finalize() throws IOException
```

抛出异常：IOExceptio，如果发生 I/O 错误。

3.8.5 getChannel 方法——返回与此文件输入流有关的惟一 FileChannel 对象

返回与此文件输入流有关的唯一 FileChannel 对象。所返回通道的初始 position 将等于到目前为止从文件中读取的字节数。从此流中读取的字节会使通道的位置递增，显式地或通过读取来更改通道的位置会更改此流的文件位置。

方法声明

```
public int available() throws IOException
```

返回值：与此文件输入流有关的文件通道。

示例

本示例创建文件字节输入流对象，并通过 getChannel()方法返回与此文件输入流有关的惟一 FileChannel 对象，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");     // 用文件路径新建一个文件对象
try {
    FileInputStream fi = new FileInputStream(file);
    Channel c = fi.getChannel();
    fi.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.8.6 getFD 方法——获取 File 对象连接的 FileDescriptor 对象

返回表示到文件系统中实际文件的连接的 FileDescriptor 对象，该文件系统正被此 FileInputStream 使用。

方法声明

```
public final FileDescriptor getFD() throws IOException
```

返回值：与此流有关的文件描述符对象。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例创建文件字节输入流对象，并通过 getFD()方法返回实际文件的连接的 FileDescriptor 对象，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    FileInputStream fi = new FileInputStream(file);
    FileDescriptor fd = fi.getFD();
    fi.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.8.7 read 方法——从文件字节流读取数组

read()方法用于从文件字节输入流中读取数据，该方法提供了多种重载形式。

方法声明 1

读取下一个字节。

```
public int read() throws IOException
```

返回值：返回下一个数据字节；如果已到达文件末尾，则返回-1。

抛出异常：IOException。

示例

本示例通过 FileInputStream 类从指定磁盘文件中读数据。

```
File file = new File("H://Getif.java");
try {
    FileInputStream fis = new FileInputStream(file);
    int length;
    while((length = fis.read())!=-1){
        System.out.println((char)length);
    }
    fis.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 2

自输入流中读取数据并存放在指定的字节数组中。

```
public int read(byte[] b) throws IOException
```

入口参数：b 是保存从流中读取的数据的 byte 数组。

返回值：返回读取的字节数，若已经到达了文件流的末尾，则返回-1。

抛出异常：IOException。

示例

本示例通过 FileInputStream 类，通过调用文件输入流的 read()方法读取指定

PDF电子书基地 http://dayo1982.400gb.com

磁盘文件中数据。

```
File file = new File("H://Getif.java");
try {
    FileInputStream fis = new FileInputStream(file);
    byte b[] = new byte[1024];
    int length ;
    while((length = fis.read(b)) != -1){
        System.out.println(new String(b));
    }
    fis.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 3

从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。

```
public int read(byte[] b,int off,int len) throws IOException
```

入口参数

b：保存从流中读取数据的 byte 数组。

off：指定 read()方法把数据存放在字节数组 b 中的索引位置。

len：指定该方法将读取的最大字节数。

返回值：返回读取的字节数，若已经到达文件流末尾，则返回-1。

抛出异常：IOException。

示例

本示例通过 FileInputStream 类，通过调用文件输入流的 read()方法读取指定磁盘文件中 10 个字节数据。

```
File file = new File("H://Getif.java");
try {
    FileInputStream fin = new FileInputStream(file);
    byte[] bs = new byte[20];
    int i = fin.read(bs, 0, 10);
    System.out.println("控制台输入的内容：" + new String(bs).trim());
    fin.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.8.8 skip 方法——跳过并丢弃 n 个字节

从输入流中跳过并丢弃 *n* 个字节的数据。

方法声明

```
public long skip(long n) throws IOException
```

返回值：实际跳过的字节数。

抛出异常：IOException：如果 *n* 为负，如果该流不支持查找操作，或者发生 I/O 错误。

说明

出于各种原因，skip 方法最终跳过的字节数可能更少一些，甚至可能为 0。如果 *n* 为负，则抛出 IOException，即使 InputStream 超类的 skip 方法在这种情况下没有执行任何操作。返回实际跳过的字节数。此方法跳过的字节可能多于底层实现文件中剩余的字节。这不会产生异常，并且跳过的字节数可能包括底层实现文件的 EOF（文件结束符）之后的一些字节数。如果试图在跳过末尾之后读取流，那么会返回-1（指示文件结束）。

示例

本示例创建文件字节输入流对象，并通过 skip()方法跳过 10 个字节然后开始读数据并输出，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    FileInputStream fi = new FileInputStream(file);
    fi.skip(10);      //跳过 10 个字节
    int i;
    while((i = fi.read()) != -1){
        System.out.print((char)i);
    }
    fi.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.9 FileOutputStream 类——文件字节输出流

java.io 包中 FileOutputStream 类是文件输出流，文件输出流是用于将数据写入 File 或 FileDescriptor 的输出流。FileOutputStream 用于写入诸如图像数据之类的原始字节的流。要写入字符流，请考虑使用 FileWriter。

3.9.1 构造方法——FileOutputStream 类的构造方法

该类构造方法如下表所示。

FileOutputStream 类的构造方法

构 造 方 法	使 用 说 明
FileOutputStream(File file)	创建一个向指定 File 对象表示的文件中写入数据的文件输出流

PDF电子书基地 http://dayo1982.400gb.com

续表

构 造 方 法	使 用 说 明
FileOutputStream(File file , boolean append)	创建一个向指定 File 对象表示的文件中写入数据的输出流
FileOutputStream(FileDescriptor filed)	创建一个向指定文件描述符处写入数据的输出文件流
FileOutputStream(String name)	创建一个向具有指定名称的文件中写入数据的输出文件流
FileOutputStream(String name,boolean append)	创建一个向具有指定 name 的文件中写入数据的输出文件流

3.9.2 close 方法——关闭文件输出流并释放资源

该方法关闭文件输出流并释放于此流相关的所有系统资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例实现向指定磁盘文件中写数据，通过调用 close()方法将流关闭。

```
File file = new File("H://Getif.java");
try {
    FileOutputStream fin = new FileOutputStream(file);
    byte b[] = "Java 编程词典".getBytes();
    fin.write(b);
    fin.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.9.3 finalize 方法——流销毁时关闭输出流

清理到文件的连接，并确保在不再引用此文件输出流时调用此流的 close 方法。由于重写了 Object 的 finalize 方法，所以在垃圾回收时会自动调用。

方法声明

```
protected void finalize() throws IOException
```

抛出异常：IOExceptio 如果发生 I/O 错误。

3.9.4 getChannel 方法——获取输出流的 FileChannel 对象

返回与此文件输出流有关的唯一 FileChannel 对象。所返回通道的初始 java.nio.channels.FileChannel#position()position 将等于到目前为止从文件中读取的字节数，除非此流处于挂起模式，在这种情况下，它将等于文件的大小。将字

节写入此流中会使通道的位置相应地增加。显式地或通过写入来更改通道的位置会更改此流的文件位置。

方法声明

```
public FileChannel getChannel()
```

返回值：与此文件输出流有关的文件通道。

示例

本示例创建文件字节输出流对象，并通过 getChannel()方法返回与此文件输出流有关的惟一 FileChannel 对象，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    FileOutputStream fos = new FileOutputStream(file);
    Channel c = fos.getChannel();
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.9.5 getFD 方法——获取 File 对象连接的 FileDescriptor 对象

返回与此流有关的文件描述符。

方法声明

```
public final FileDescriptor getFD() throws IOException
```

返回值：表示到文件系统中的某个文件的连接的 FileDescriptor 对象，该文件系统正被此 FileOutputStream 对象使用。

抛出异常：IOException：如果发生 I/O 错误。

示例

本示例创建文件字节输出流对象，并通过 getFD()方法返回实际文件的连接的 FileDescriptor 对象，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    FileOutputStream fos = new FileOutputStream(file);
    FileDescriptor fd = fos.getFD();
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.9.6 write 方法——向文件输出流写数据

该方法向指定文件输出流中写入数据。该方法提供了多种重载形式。

PDF电子书基地 http://dayo1982.400gb.com

方法声明 1

向文件输出流中写入一个字节的数据。

```
public void write(int b) throws IOException
```

入口参数：b 是向流中写入的字节。

抛出异常：IOException。

示例

本示例实现向指定磁盘文件中写数据，并调用文件输出流的 write()方法向文件写数据，write()方法给定的参数为小写字母“a”、“b”、“c”对应的 ASCII 值。

```
File file = new File("H://Getif.java");
try {
    FileOutputStream fin = new FileOutputStream(file);
    fin.write(97);
    fin.write(98);
    fin.write(99);
    fin.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 2

将 b.length 个字节从指定 byte 数组写入此文件输出流中。

```
public void write(byte[] b) throws IOException
```

入口参数：b 为要写入流的字节数组。

抛出异常：IOException。

示例

本示例实现从 12.txt 文件中读取 1024 个字节数据，将其写入 Getif.java 文件中。

```
File file = new File("H://Getif.java");
File file2 = new File("H://12.txt");
try {
    byte[] by = new byte[1024];
    FileOutputStream fin = new FileOutputStream(file);
    FileInputStream fon = new FileInputStream(file2);
    fon.read(by);
    fin.write(by);
    fon.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 3

该方法从给定字节数组中起始于偏移量 off 处写 len 个字节到输出流。

```
public void write(byte[] b,int off,int len) throws IOException
```

参数说明

b：存放了数据的字节数组。

off：数据的起始偏移量。

len：要输出的字节数。

抛出异常：IOException。

示例

本示例实现将控制台中输入的内容，写到指定磁盘文件中。

```
File file = new File("H://Getif.java");
try {
    byte[] by = new byte[1024];
    FileOutputStream fin = new FileOutputStream(file);
    System.out.println("输入一行文件，并存入磁盘");
    int b= System.in.read(by);
    fin.write(by,0,b);
    fin.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

示例

本示例定义类的运行次数、运行时间、操作系统名称等属性，在主方法中调用 loadConfig()方法从 myProgram.cfg 配置文件中获取程序上次运行时的属性信息。如果没有配置文件或者配置文件中没有内容，将在控制台输出“这是程序第一次运行，其他信息还没有初始化。”的信息，否则输出程序上次运行时定义的属性信息，并且将本次程序运行的属性通过调用 putConfig()方法保存到配置文件中。

```
private static String filePath = "./myProgram.cfg";
private static File file = new File(filePath);
private static int runCount = 0;
private static String date = String.format("%tF %<tT", new Date());
private static String os = System.getProperty("os.name");
private static String dataStr = "";
public static void main(String[] args) {
    loadConfig();
if (dataStr.isEmpty())
    System.out.println("这是程序第一次运行,其他信息还没有初始化。");
else {
    System.out.println("程序已经运行了" + runCount + "次了");
    System.out.println("程序上次运行时间是：" + date);
    System.out.println("上次运行程序的操作系统是：" + os);
}
putConfig();
```


PDF电子书基地 http://dayo1982.400gb.com

```
}
private static void putConfig() {
        String dataStr;
        try {
                if (!file.exists())
                        file.createNewFile();
                String date2 = String.format("%tF %<tT", new Date());
                dataStr = (runCount + 1) + "," + date2 + "," + os;
                byte[] data = dataStr.getBytes();
                FileOutputStream fout = new FileOutputStream(file);
                fout.write(data);
                fout.flush();
                fout.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
}
private static void loadConfig() {
        try {
                if (!file.exists())
                        file.createNewFile();
                byte[] data = new byte[64];
                FileInputStream fis = new FileInputStream(file);
                int rs = 0;
                while ((rs = fis.read(data)) > 0) {
                        dataStr += new String(data, 0, rs);
                }
                if (!dataStr.isEmpty()) {
                        String[] sets = dataStr.split(",");
                        runCount = Integer.parseInt(sets[0]);
                        date = sets[1];
                        os = sets[2];
                }
                fis.close();
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

3.10 FileReader 类——文件的字符输入流

java.io 包中 FileReader 类继承自 Reader 类，可以从指定路径中读取字节数据。提供了实现读取字符输入流的各种方法。通过使用 FileReader 类可以避免在读取字符时出现的乱码现象。

3.10.1 构造方法——FileReader 类的构造方法

FileReader 类的构造方法如下表所示。

FileReader 类的构造方法

构 造 方 法	使 用 说 明
FileReader(File file)	在给定读取数据的 File 对性创建 FileReader 实例
FileReader(FileDescriptor fd)	在给定从中读取数据的 FileDescriptor 创建 FileReader 实例
FileReader(String name)	在给定从中读取数据的文件名的情况下创建一个新 FileReader 实例

3.10.2 close 方法——关闭字符输入流

该方法关闭流并释放与之关联的所有资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例通过 FileReader 类实现获取 H 盘中 Getif.java 文件中信息，并将其内容在控制台中输出。调用 close()方法将流关闭。

```
File file = new File("H://Getif.java");
try {
    FileReader fReader=new FileReader(file);
    char[] data=new char[512];
    int rs=0;
    while((rs=fReader.read(data))>0){
        System.out.println(new String(data,0,rs));
    }
    fReader.close()
} catch (Exception e) {
    e.printStackTrace();
}
```

3.10.3 read 方法——读取字符数据

read()方法从字符输入流中读取字符数据，该方法提供了多种重载形式。

方法声明 1

该方法从输入流中读取单个字符。

```
public int read() throws IOException
```

抛出异常：IOException。

示例

本示例创建 FileReader 对象，并通过 read()方法每次读取指定磁盘文件中一个字符数据。通过 while 循环将指定磁盘文件中数据全部输出在控制台上。

```
File file = new File("H://Getif.java");
try {
```


PDF电子书基地 http://dayo1982.400gb.com

```
        FileReader fReader=new FileReader(file);
        int length;
        while((length = fReader.read())!=-1){
            System.out.println((char)length);
        }
        fReader.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

方法声明 2

```
public int read(char[] ch) throws IOException
```

入口参数：ch 自输入流读取字符数组 ch 的长度的字符，将其放置在字符数组 ch 中。

返回值：返回读取的字符数，如果已到达流的末尾，则返回-1。

抛出异常：IOException。

示例

本示例从指定磁盘文件中读取 1024 长度的字符，并在控制台上输出。

```
File file = new File("H://Getif.java");
try {
    FileReader fReader = new FileReader(file);
    char[] ch = new char[1024];
    int length = fReader.read(ch);
    System.out.println(new String(ch, 0, length));
    fReader.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 3

读取 len 个字节，放置到以下标 off 开始的字节数组 ch 中。

```
public int read(char[] ch,int off,int len) throws IOException
```

入口参数

ch：read()方法把数据存放到的字符数组。

off：read()方法把数据存放到字符数组的索引位置。

len：指定该方法将读取的最大字符数。

返回值：返回读取的字符数，如果已到达流的末尾，则返回-1。

抛出异常：IOException。

示例

本示例创建 FileReader 类对象，通过调用 FileReader 类的 read()方法实现每次从指定磁盘文件中读取 10 个字符的数据，至文件读取完毕。

```
File file = new File("H://Getif.java");
try {
    FileReader fReader = new FileReader(file);
    char[] ch = new char[10];
    int len;
    while ((len = fReader.read(ch, 0, 10)) != -1) {
        String str = new String(ch, 0, len);
        System.out.println(str);
    }
    fReader.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.11 FileWriter 类——文件的字符输出流

java.io 包中的 FileWriter 类是文件字符输出流，该类继承自 Writer 类，用来写入字符文件的便捷类。文件字符输出流关联指定路径的文件，数据通过文件字符输出流以字符为单位输出并保存到文件中。

3.11.1 构造方法——FileWriter 类的构造方法

该类的构造方法如下表所示。

FileWriter 类的构造方法

构 造 方 法	使 用 说 明
FileWriter(File file)	根据跟定的 File 对象构造一个 FileWriter 对象
FileWriter(String name)	根据给定的文件名构造一个 FileWriter 对象
FileWriter(String name , boolean bool)	根据给定的文件名以及指示是否附加写入数据的布尔值创建 FileWriter

3.11.2 close 方法——关闭字符输出流

该方法关闭流并释放与之关联的所有资源。

方法声明

```
public void close() throws IOException
```

抛出异常： IOException。

示例

本示例创建 FileWriter 对象，实现向指定磁盘文件中写数据，最后通过调用 close()方法将流关闭。

```
File file = new File("H://Getif.java");
try {
    String filePath = "通过字符文件流向文件写数据";
```


PDF电子书基地 http://dayo1982.400gb.com

```
        FileWriter fWriter = new FileWriter(file);
        fWriter.write(filePath);
        fWriter.flush();
        fWriter.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.11.3 flush 方法——刷新缓冲区

刷新输出流，并输出所有被缓存的字符。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException。

示例

本示例创建 FileWriter 对象，实现将字符串数组中的各个元素写入指定磁盘文件中。通过 flush()方法刷新流。

```
File file = new File("H://Getif.java");
try {
    String filePath[] = {"Java 编程词典","Java 编程专家","Java 经典"};
    FileWriter fWriter = new FileWriter(file);
    for(int i = 0;i<filePath.length;i++){
        fWriter.write(filePath[i]);
    }
    fWriter.flush();
    fWriter.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.11.4 write 方法——向字符输出流写数据

write()方法实现向输出流中写入字符，该方法提供了多种重载形式。

方法声明 1

```
public void write(int i) throws IOException
```

入口参数：i 为要写入流的 int 类型参数。

抛出异常：IOException。

示例

本示例创建 FileWriter 对象，通过 FileWriter 类的 write()方法将小写字母“a”～“z”写入指定的磁盘文件中。

```
File file = new File("H://Getif.java");
try {
```

```
        FileWriter fWriter = new FileWriter(file);
        int a = (int)'a';
        int z = (int)'z';
        for(int i = a;i<=z;i++){
            fWriter.write(i);
        }
        fWriter.flush();
        fWriter.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

方法声明 2

从给定的字符数组中开始偏移量 off 处写 len 个字符到输出流。

```
public void write(char[] ch,int off,int len) throws IOException
```

入口参数

ch：存放了数据的字符数组。

off：开始写入字符处的字符数组的索引。

len：写入字符数组的字符数量。

抛出异常：IOException。

示例

本示例创建 FileWriter 对象，并通过 FileWriter 类的 write()方法向指定磁盘文件中写数据。

```
File file = new File("H://Getif.java");
try {
    FileWriter fWriter = new FileWriter(file);
    char[] ch = "Java 编程词典".toCharArray();
    fWriter.write(ch,0,ch.length);
    fWriter.flush();
    fWriter.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

方法声明 3

将指定字符串写入流。

```
public void write(String str) throws IOException
```

入口参数：str 为要写入流的字符串。

抛出异常：IOException。

示例

本示例创建 FileWriter 对象，并通过 FileWriter 类的 write()方法将字符串写

PDF电子书基地 http://dayo1982.400gb.com

入指定磁盘文件中。

```
File file = new File("H://Getif.java");
try {
    FileWriter fWriter = new FileWriter(file);
    String str = "Java 编程词典速查手册";
    fWriter.write(str);
    fWriter.flush();
    fWriter.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.12 InputStream 类——字节输入流

java.io 包中的 InputStream 类是字节输入流的抽象类，定义了操作输入流的各种方法。

3.12.1 available 方法——获取有效的可读取的字节数

返回此输入流下一个方法调用可以不受阻塞地从此输入流读取（或跳过）的估计字节数。下一个调用可能是同一个线程，也可能是另一个线程。一次读取或跳过此估计数个字节不会受阻塞，但读取或跳过的字节数可能小于该数。

方法声明

```
public int available() throws IOException
```

返回值：可以不受阻塞地从此输入流中读取（或跳过）的估计剩余字节数。

抛出异常：IOException，如果此文件输入流已通过调用 close 关闭，或者发生 I/O 错误。

注意

有些 InputStream 的实现将返回流中的字节总数，但也有很多实现不会这样做。试图使用此方法的返回值分配缓冲区，以保存此流所有数据的做法是不正确的。如果已经调用 close()方法关闭了此输入流，那么此方法的子类实现可以选择抛出 IOException。类 InputStream 的 available 方法总是返回 0。

示例

本示例创建字节输入流对象，并通过 available()方法估计了剩余的字节数，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    InputStream in = new FileInputStream(file);
    System.out.println("文件大小为" + file.length() + "Bytes");
    in.read();          //读取一个字节
```

```
        System.out.println("估计文件还有" + in.available() + "Bytes");
        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.12.2 close 方法——关闭输入流

关闭此输入流并释放与该流关联的所有系统资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例通过 System.in 实例化 InputStream 类，实现输出控制台上输入的信息，最后将流关闭。

```
InputStream is=System.in;
try {
    byte[] bs=new byte[50];
    is.read(bs);
    System.out.println("控制台输入的内容："+new String(bs).trim());
    is.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.12.3 mark 方法——在此输入流中标记当前的位置

在此输入流中标记当前的位置。对 reset 方法的后续调用会在最后标记的位置重新定位此流，以便后续读取重新读取相同的字节。

方法声明

```
public void mark(int readlimit)
```

入口参数：readlimit 为在标记位置失效前可以读取字节的最大限制。

示例

本示例创建字节输入流对象，实现循环读取指定磁盘文件中前 5 个字节。

```
try {
File file = new File("H://Getif.java"); // 用文件路径新建一个文件对象
//创建输入流
InputStream bip = new BufferedInputStream(new FileInputStream(file));
int count = 0;
bip.mark(50);
for (int i = 0; i < file.length(); i++) {
    int read = bip.read();
    if (count % 5 == 0) {
```


PDF电子书基地 http://dayo1982.400gb.com

```
                bip.reset();
                System.out.println();
            }
            count++;
            System.out.print((char) read);
        }
        bip.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.12.4 markSupported 方法——判断是否支持 mark 和 reset 方法

测试此输入流是否支持 mark 和 reset 方法。是否支持 mark 和 reset 是特定输入流实例的不变属性。InputStream 的 markSupported 方法返回 false。

方法声明

```
public boolean markSupported()
```

返回值：如果此输入流实例支持 mark 和 reset 方法，则返回 true，否则返回 false。

示例

本示例创建字节输入流对象，并通过 markSupported()方法输出了，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java"); // 用文件路径新建一个文件对象
try {
    InputStream in = new FileInputStream(file);
    if(in.markSupported()){
        System.out.println("支持 mark 和 reset 方法!");
    }
    in.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.12.5 read 方法——读取字节输入流中的数据

该方法用于读取字节输入流中的数据。read()方法存在多种重载形式。

方法声明 1

以整数形式返回实际读取的字节数。

```
public int read(byte[] byte) throws IOException
```

入口参数：byte 指定读取的字节数组。

抛出异常：IOException。

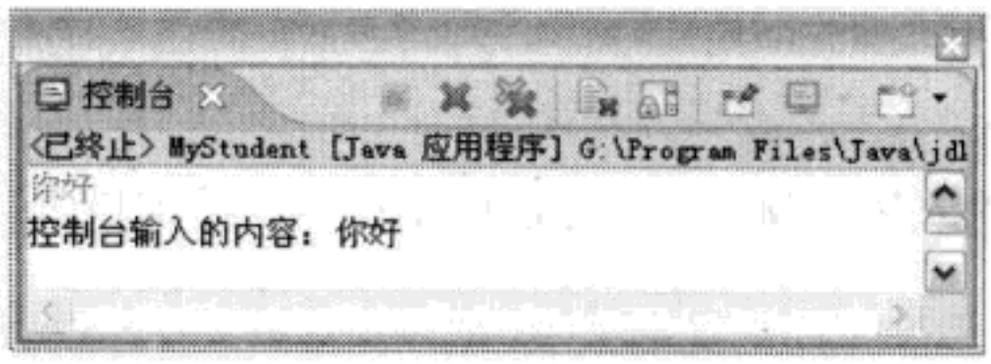
示例

本示例创建 InputStream 类实例 is，并为赋值为 System 类的 in 属性，该属性

是控制台输入流，从 is 输入流中获取一些字节信息，用这些字节信息创建一个字符串，并输出到控制台中。

```
InputStream is=System.in;
    try {
        byte[] bs=new byte[50];
        int i = is.read(bs);
                System.out.println("控制台输入的内容："+new String(bs).trim());
         is.close();
    } catch (IOException e) {
            e.printStackTrace();
        }
```

运行程序在控制台中输出字符串“你好”，控制台中将输出字符串“控制台输入的内容：“你好”，如下图所示。



获取控制台输入信息

方法声明 2

该方法读取 len 个字节，放置在以下标 off 开始的字节数组中，返回实际读取的字节的数量。

```
public int read(byte[] byte,int off,int len) throws IOException
```

入口参数

byte：读取数据的缓冲区。

off：数组 byte 中将写入数据的初始偏移量。

len：要读取的最大字节数。

返回值：如果 byte 的长度为 0，则不读取任何字节并返回 0；如果因为流位于文件末尾而没有可用的字节，则返回值是-1；否则，至少读取一个字节并将其储存在 byte 中。

抛出异常：IOException。

示例

本示例创建 InputStream 类实例 is，并将其赋值为 System 类的 in 属性，实现将控制台中输出信息，重新显示在控制台上，并通过 read()方法控制输出的字节数。

```
InputStream is = System.in;
    try {
        byte[] bs = new byte[50];
```


PDF电子书基地 http://dayo1982.400gb.com

```
            int i = is.read(bs, 0, 10);
            System.out.println("控制台输入的内容： " + new String(bs).trim());
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

3.12.6 reset 方法——重新定位到最后一次 mark()方法时的位置

将输入流重新定位到最后一次对此输入流调用 mark()方法时的位置。

方法声明

```
public void reset() throws IOException
```

抛出异常： IOException。

示例

本示例创建字节输入流对象，实现循环读取指定磁盘文件中前 5 个字节。

```
try {
        File file = new File("H://Getif.java");
        FileInputStream fin = new FileInputStream(file);
        BufferedInputStream bip = new BufferedInputStream(fin);
        int count = 0;
        bip.mark(50);
        for (int i = 0; i < file.length(); i++) {
            count++;
            int read = bip.read();
            if (count == 5)
                bip.reset();
            System.out.println((char) read);
        }
        bip.close();
    } catch (Exception e) {
        e.printStackTrace();
```

3.12.7 skip 方法——跳过并丢弃 *n* 个字节

从输入流中跳过并丢弃 *n* 个字节的数据。

方法声明

```
public long skip(long n) throws IOException
```

返回值：实际跳过的字节数。

抛出异常：IOException：如果 *n* 为负，如果该流不支持查找操作，或者发生 I/O 错误。

说明

出于各种原因，skip 方法结束时跳过的字节数可能小于该数，也可能为 0。导致这种情况的原因很多，跳过 *n* 个字节之前已到达文件末尾只是其中一种可

能。返回跳过的实际字节数。如果 *n* 为负，则不跳过任何字节。此类的 skip 方法创建一个 byte 数组，然后重复将字节读入其中，直到读够 *n* 个字节或已到达流末尾为止。建议子类提供此方法更为有效的实现。例如，可依赖搜索能力的实现。

示例

本示例创建字节输入流对象，并通过 skip()方法跳过 10 个字节然后开始读数据并输出，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");     // 用文件路径新建一个文件对象
try {
    InputStream in = new FileInputStream(file);
    in.skip(10);      //跳过 10 个字节
    int i;
    while((i = in.read()) != -1){
        System.out.print((char)i);
    }
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.13 OutputStream 类——字节输出流

java.io 包中的 OutputStream 抽象类是表示输出字节流的所有类的超类。定义了输出流的各种操作方法。

3.13.1 close 方法——关闭字节输出流

关闭输出流并释放与此流有关的所有系统资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException。

示例

本示例创建 OutputStream 类实例 out，并实现向控制台上输出信息，最后将流关闭。

```
OutputStream out=System.out;
try {
    byte[] bs="本实例使用 OutputStream 输出流，在控制台输出字符串\n".getBytes();
    out.write(bs);
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.13.2 flush 方法——刷新缓冲区

刷新此输出流并强制写出所有缓冲的输出字节。flush 的常规协定是：如果此输出流的实现已经缓冲了以前写入的任何字节，则调用此方法指示应将这些字节立即写入它们预期的目标。如果此流的预期目标是由基础操作系统提供的一个抽象（如一个文件），则刷新此流只能保证将以前写入到流的字节传递给操作系统进行写入，但不保证能将这些字节实际写入到物理设备（如磁盘驱动器）。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例创建 OutputStream 类实例 out，并实现向控制台上输出信息。调用 flush()方法刷新此输出流并强制写出所有缓冲的输出字节，最后将流关闭。

```
OutputStream out=System.out;
try {
    byte[] bs="本实例使用 OutputStream 输出流，在控制台输出字符串\n".getBytes();
    out.write(bs);
    out.flush();
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.13.3 write 方法——向输出流写数据

该方法用于向输出流中写数据。write()方法提供了多种重载形式。

方法声明 1

该方法将参数转换为字节，然后写到输出流。

```
public abstract void write(int b) throws IOException
```

抛出异常：IOException。

示例

本示例创建 OutputStream 类实例 out，并通过 write()方法向流中写入字节 97。由于字节 97 对应的 ASCII 码值是小写字母“a”，因此控制台上输出的是小写字母“a”。

```
OutputStream out=System.out;
try {
        out.write(97);
    out.close();
} catch (IOException e) {
```

```
        e.printStackTrace();
    }
```

方法声明 2

```
public void write(byte[] byte) throws IOException
```

入口参数：byte 是指定写入流的字节数组。

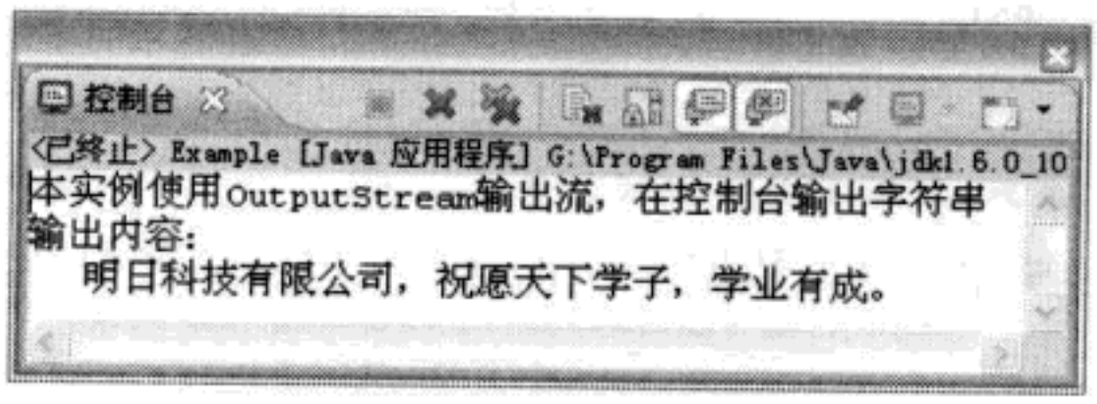
抛出异常：IOException。

示例

本示例创建 OutputStream 类实例 out，通过 System.out 实现标准输出流。并通过 write()方法向流中写数据。

```
OutputStream out=System.out;
try {
    byte[] bs="本实例使用 OutputStream 输出流，在控制台输出字符串\n".getBytes();
    out.write(bs);
    bs="输出内容：\n".getBytes();
    out.write(bs);
    bs="        明日科技有限公司，祝愿天下学子，学业有成。".getBytes();
    out.write(bs);
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

本实例的运行结果如下图所示。



实现向控制台输出信息

方法声明 3

```
public void write(byte[] byte,int off,int len) throws IOException
```

入口参数

byte：指定写入流的数据。

off：要写入流的数组中数据的起始索引位置。

len：要写入流的字节数。

抛出异常：IOException。

示例

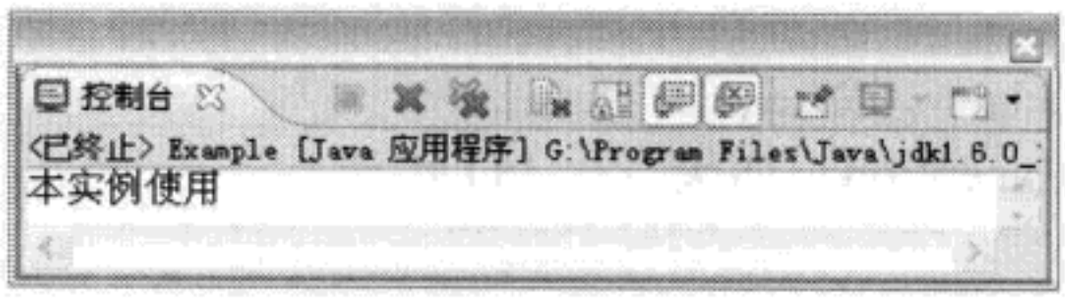
本示例创建 OutputStream 类实例 out，通过 System.out 实现标准输出流，并

PDF电子书基地 http://dayo1982.400gb.com

通过 write()方法限制写入流的字节数，最后将流关闭。

```
OutputStream out=System.out;
try {
    byte[] bs="本实例使用 OutputStream 输出流，在控制台输出字符串\n".getBytes();
    out.write(bs,0,10);
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

由于汉字占两个字节。因此控制台上输出“本实例使用”。运行结果如下图所示。



限制输出到控制台上的数据长度

3.14 ObjectOutputStream 类——对象输出流

java.io 包中的 ObjectOutputStream 类是对象输出流类，该类提供了将对象写入流的相应方法。但只有支持 Serializable 接口的对象才能写入流中。

3.14.1 构造方法——ObjectOutputStream 类的构造方法

ObjectOutputStream 类的构造方法如下表所示。

ObjectOutputStream 类的构造方法

构 造 方 法	使 用 说 明
ObjectOutputStream()	创建一个使用私用数据实现 ObjextOutputStream 类
ObjectOutputStream(OutputStream out)	创建写入指定 OutputStream 的 ObjectOutputStream

3.14.2 close 方法——关闭对象输出流

关闭流。必须调用此方法来释放与此流有关的所有资源。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例创建 ObjectOutputStream 实例，通过 close()方法将流关闭。

```
try {
    File file = new File("H:\\obj.dat");
```

```
        FileOutputStream fos = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.close();
        fos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

3.14.3 flush 方法——刷新缓冲区

刷新该流的缓冲。此操作将写入所有已缓冲的输出字节，并将它们刷新到底层流中。

方法声明

```
public void flush() throws IOException
```

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例创建 ObjectOutputStream 实例，通过 close()方法将流关闭。

```
try {
    File file = new File("H:\\obj.dat");
    FileOutputStream fos = new FileOutputStream(file);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.write(10);
    oos.flush();     //刷新该流的缓冲
    oos.close();
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.14.4 writeObject 方法——向输出流写入对象

该方法将指定的对象写入 ObjectOutputStream。

方法声明

```
public final void writeObject(Object obj) throws IOException
```

入口参数：obj 是要写入的对象。

抛出异常：IOException。

示例

本示例创建类 Student，该类实现 Serializable 接口，代码如下。

```
public class Student implements Serializable {
    String name;
    String sex;
}
```

通过 ObjectOutputStream 类将对象写入文件中。

PDF电子书基地 http://dayo1982.400gb.com

```
File file = new File("H://Getif.java");
try {
    FileOutputStream fop = new FileOutputStream(file);
    ObjectOutputStream ops = new ObjectOutputStream(fop);
    Student stu = new Student();
    stu.name = "李四";
    stu.sex = "男";
    ops.writeObject(stu);
    ops.close();
    fop.close();
} catch (Exception e) {
        e.printStackTrace();
}
```

3.14.5 writeBytes 方法——向输出流写入字符串

以字节序列形式写入一个字符串对象。

方法声明

```
public void writeBytes(String str) throws IOException
```

入口参数：str 是要写入的字符串对象。

抛出异常：IOException。

示例

本示例通过 ObjectOutputStream 类的 writeBytes()方法将字符串写入指定的系统文件中。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
ObjectOutputStream ops = new ObjectOutputStream(fin);
String str = "Java 编程词典";
ops.writeBytes(str);
ops.close();
```

3.14.6 write 方法——向输出流写入 byte 值

向对象输出流中写入一个 byte 数组。

方法声明 1

```
public void write(byte[] bf) throws IOException
```

入口参数：bf 为要写入的数据。

抛出异常：IOException。

示例

本示例使用 ObjectOutputStream 类将字节数组中的元素写入到指定磁盘文件中。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
```

```
ObjectOutputStream ops = new ObjectOutputStream(fin);
byte[] by = "Java 编程词典".getBytes();
ops.write(by);
ops.close();
```

方法声明 2

向对象输出流中写入字节的子数组。

```
public void write(byte[] buf, int off, int len) throws IOException
```

入口参数：

buf：要写入的数据。

off：数据中的初始偏移量。

len：写入的字节数。

抛出异常：IOException 如果发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将字节数组中的元素（初始偏移量为 0，字节数为 3）写入指定磁盘文件中。

```
try {
    File file = new File("H:\\object\\int.dat");
    FileOutputStream fos = new FileOutputStream(file);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    byte[] bs = "abcd".getBytes();
    oos.write(bs, 0, 3);
    oos.flush();
    oos.close();
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.14.7 writeChar 方法——向输出流写入 char 值

写入一个 16 位的 char 值。

方法声明

```
public void writeChar(int val) throws IOException
```

入口参数：val 为要写入的 char 值。

抛出异常：如果在将数据写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将字符写入指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
```


PDF电子书基地 http://dayo1982.400gb.com

```
char c = 'a';
oos.writeChar(c);
```

3.14.8 writeChars 方法——向输出流写入 String 值

以 char 序列形式写入一个 String。

方法声明

```
public void writeChar(int val) throws IOException
```

入口参数：str 为要写入的 char 的 String。

抛出异常：如果在将数据写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将字符串写入到指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
String s = "明日科技编程词典";
oos.writeChars(s);
```

3.14.9 writeDouble 方法——向输出流写入 double 值

写入一个 64 位的 double 值。

方法声明

```
public void writeDouble(double val) throws IOException
```

入口参数：val 为要写入的 double 值。

抛出异常：IOExeption 如果在将数据写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将 double 值写入到指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
double d = 123.456;
oos.writeDouble(d);
      oos.flush();
```

3.14.10 writeFields 方法——将已缓冲的字段写入流中

将已缓冲的字段写入流中。

方法声明

```
public void writeFields() throws IOException
```

抛出异常

IOException：如果在写入底层流时发生 I/O 错误。

NotActiveException：未调用类 writeObject 方法来写入对象的状态时调用。

示例

本示例使用 ObjectOutputStream 类将已缓冲的字段写入流中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
double d = 123.456;
oos.writeDouble(d);
oos.writeFields ();
oos.close();
fos.close();
```

3.14.11 writeFloat 方法——向输出流写入 float 值

向对象输出流中写入一个 32 位的 float 值。

方法声明

```
public void writeFloat(float val) throws IOException
```

入口参数：val 为要写入的 float 值。

抛出异常：IOException。

示例

本示例使用 ObjectOutputStream 类将指定的 float 值写入系统文件中。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
ObjectOutputStream ops = new ObjectOutputStream(fin);
ops.writeFloat(100.123f);
ops.close();
```

3.14.12 writeInt 方法——向输出流写入 int 值

向对象输出流中写入一个 32 为的 int 值。

方法声明

```
public void writeInt(int val) throws IOException
```

入口参数：val 为要写入的整数值。

抛出异常：IOException。

示例

本示例使用 ObjectOutputStream 类将指定的 int 值写入系统文件中。

```
File file = new File("H://Getif.java");
FileOutputStream fin = new FileOutputStream(file);
ObjectOutputStream ops = new ObjectOutputStream(fin);
ops.writeInt(100);
ops.close();
```


PDF电子书基地 http://dayo1982.400gb.com

3.14.13 writeLong 方法——向输出流写入 long 值

写入一个 64 位的 long 值。

方法声明

```
public void writeLong(long val) throws IOException
```

入口参数：val 为要写入的 long 值。

抛出异常

IOException：如果在写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将 long 值写入指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
long v = 123;
oos.writeLong(v);
oos.writeFields ();
oos.close();
fos.close();
```

3.14.14 writeShort 方法——向输出流写入 short 值

写入一个 16 位的 short 值。

方法声明

```
public void writeShort(int val) throws IOException
```

入口参数：val 为要写入的 short 值。

抛出异常

IOException：如果在写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将 short 值写入指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
short v = 123;
oos.writeShort(v);
oos.writeFields ();
oos.close();
fos.close();
```

3.14.15 writeUTF 方法——向输出流写入 float 值

以 UTF-8 修改版格式写入此 String 的基本数据。注意，将 String 作为基本数

据写入流中与将它作为 Object 写入流中明显不同。由 writeObject 写入的 String 实例最初是作为 String 写入流中的。然后，writeObject()调用将对该字符串的引用写入流中。

方法声明

```
public void writeUTF(String str) throws IOException
```

入口参数：str 为要写入的 String。

抛出异常

IOException：如果在写入底层流时发生 I/O 错误。

示例

本示例使用 ObjectOutputStream 类将 short 值写入指定磁盘文件中。

```
File file = new File("H:\\object\\int.dat");
FileOutputStream fos = new FileOutputStream(file);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeUTF("abcd");
oos.flush();
oos.close();
fos.close();
```

3.15 ObjectInputStream 类——对象输入流

java.io 包中的 ObjectInputStream 类，实现了从流中读取对象的方法。只有支持 java.io.Serializable 接口的对象才能从流读取。

3.15.1 构造方法——ObjectInputStream 类的构造方法

OjectInputStream 类的构造方法如下表所示。

ObjectInputStream 类的构造方法

构 造 方 法	使 用 说 明
ObjectInputStream()	为重新实现 ObjectInputStream 的子类提供一种方式
ObjectInputStream(InputStream in)	创建从指定 InputStream 读取 ObjectInputStream

3.15.2 readObject 方法——从输入流读取对象

从 ObjectInputStream 读取对象。

方法声明

```
public final Object readObject() throws IOException,ClassNotFoundException
```

返回：从流中读取的对象。

抛出异常：IOException,ClassNotFoundException。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例通过 ObjectInputStream 类中将磁盘文件中对象信息读取出来，并输出在控制台上。

```
File file = new File("H://Getif.java");
try {
    FileInputStream fism = new FileInputStream(file);
    ObjectInputStream opss = new ObjectInputStream(fism);
    Student st = (Student)opss.readObject();
    System.out.println(st.name);
    System.out.println(st.sex);
    opss.close();
    fism.close();
} catch (Exception e) {
      e.printStackTrace();
}
```

3.15.3 readInt 方法——从输入流读取 int 值

从 ObjectInputStream 读取一个 32 位 int 值。

方法声明

```
public int readInt() throws IOException
```

返回：读取的 32 位整数。

抛出异常：IOException。

示例

本示例通过 ObjectInputStream 类中从指定的磁盘文件中读取 32 位整数，将其输出到控制台上。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
ObjectInputStream ojs = new ObjectInputStream(fis);
System.out.println(ojs.readInt());
ojs.close();
```

3.15.4 readFloat 方法——从输入流读取 float 值

从 ObjectInputStream 读取一个 32 位 float 值。

方法声明

```
public float readFloat() throws IOException
```

返回：读取的 32 位 float 值。

抛出异常：IOException。

示例

本示例通过 ObjectInputStream 类中从指定的磁盘文件中读取 32 位 float，将

其输出到控制台上。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
ObjectInputStream ojs = new ObjectInputStream(fis);
System.out.println(ojs.readFloat());
ojs.close();
```

3.15.5 readchar 方法——从输入流读取 char 值

从 ObjectInputStream 读取一个 16 位 char 值。

方法声明

```
public char readchar() throws IOException
```

返回：读取的 16 位 char 值。

抛出异常：IOException。

示例

本示例通过ObjectInputStream类中从指定的磁盘文件中读取16位的char值，将其输出到控制台上。

```
File file = new File("H://Getif.java");
FileInputStream fis = new FileInputStream(file);
ObjectInputStream ojs = new ObjectInputStream(fis);
System.out.println(ojs.readChar());
ojs.close();
```

3.16 RandomAccessFile 类——随机文件访问类

java.io 包中的 RandomAccessFile 类是随机文件访问类，支持对文件的随机读写操作。随机访问文件可以读写其存储在任意位置的数据文件。

RandomAccessFile 类既不是 InputStream 类的子类，也不是 OutputStream 的子类。RandomAccessFile 类创建的流的指向可以作为源也可以作为目的地。

3.16.1 构造方法——RandomAccessFile 类的构造方法

RandomAccessFile 类的构造方法如下表所示。

RandomAccessFile 类的构造方法

构 造 方 法	使 用 说 明
RandomAccessFile(File file,String name)	创建向指定文件中读取和写入随机访问的文件流
RandomAccessFile(String name,String mode)	创建向指定文件名称中读取和写入随机访问文件流

3.16.2 close 方法——关闭数据流

关闭此随机访问文件流并释放与该流关联的所有系统资源。关闭的随机访问文件不能执行输入或输出操作，而且不能重新打开。如果此文件具有一个关联的

PDF电子书基地 http://dayo1982.400gb.com

通道，那么该通道也会被关闭。

方法声明

```
public void close() throws IOException
```

抛出异常：IOException 如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，读取了一个字节，然后关闭该流。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.readLine();
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.3 getChannel 方法——返回此文件的 FileChannel 对象

返回与此文件关联的惟一 FileChannel 对象。

方法声明

```
public FileChannel getChannel()
```

返回值：与此文件关联的文件通道。

示例

本示例按读写方式打开一个随机访问文件流，并通过 getChannel()方法返回与此文件输出流有关的惟一 FileChannel 对象，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    Channel c = r.getChannel();
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.4 getFD 方法——返回此流的不透明文件描述符对象

返回与此流关联的不透明文件描述符对象。

方法声明

```
public final FileDescriptor getFD() throws IOException
```

返回值：与此流关联的不透明文件描述符对象。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 getFD()方法返回实际文件的连接的 FileDescriptor 对象，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    FileDescriptor fd = r.getFD();
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.5 getFilePointer 方法——返回文件中的当前偏移量

返回此文件中的当前偏移量。

方法声明

```
public long getFilePointer() throws IOException
```

返回值：到此文件开头的偏移量（以字节为单位），在该位置发生下一个读取或写入操作。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 getFilePointer()方法返回当前偏移量并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(10);
    System.out.println(r.getFilePointer());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.6 length 方法——返回此文件的长度

返回此文件的长度。

方法声明

```
public long length() throws IOException
```

返回值：按字节测量的此文件的长度。

抛出异常：IOException，如果发生 I/O 错误。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例按读写方式打开一个随机访问文件流，并通过 length()方法输出了文件的长度，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r.length());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.7 read 方法——读取一个数据字节

从此文件中读取一个数据字节。以整数形式返回此字节，范围在 0～255（0x00-0x0ff）。如果尚无输入可用，将阻塞此方法。尽管 RandomAccessFile 不是 InputStream 的子类，但此方法的行为与 InputStream 的 InputStream.read()方法完全一样。

方法声明 1

```
public int read() throws IOException
```

返回值：下一个数据字节，如果已到达文件的末尾，则返回-1。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 read()方法读取了文件中的所有字节并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    int i;
    while((i = r.read()) != -1){
        System.out.println(i);
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

将最多 b.length 个数据字节从此文件读入 byte 数组。在至少一个输入字节可用前，此方法一直阻塞。

尽管 RandomAccessFile 不是 InputStream 的子类，但此方法的行为与

InputStream 的 InputStream.read(byte[])方法完全一样。

```
public int read(byte[] b) throws IOException
```

入口参数：b 为将数据读入的缓冲区。

返回值：读入缓冲区的总字节数，如果由于已到达此文件的末尾而不再有数据，则返回–1。

抛出异常

IOException：如果由于文件结束之外的某种原因不能读取第一个字节，或者随机访问文件已关闭，或者发生其他 I/O 错误。

NullPointerException：如果 b 为 null。

示例

本示例按读写方式打开一个随机访问文件流，并通过 read()方法读取了 b.length 个数据字节并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte[] bs = new byte[10];
    r.read(bs);
    for (int i = 0; i < bs.length; i++) {
        System.out.println(bs[i]);
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 3

将最多 len 个数据字节从此文件读入 byte 数组。在至少一个输入字节可用前，此方法一直阻塞。

尽管 RandomAccessFile 不是 InputStream 的子类，但此方法的行为与 InputStream 的 InputStream.read(byte[],int,int)方法完全一样。

```
public int read(byte[] b, int off, int len) throws IOException
```

入口参数

b：读入数据的缓冲区。

off：写入数据的数组 b 中的初始偏移量。

len：读取的最多字节数。

返回值：读入缓冲区的总字节数，如果由于已到达此文件的末尾而不再有数据，则返回–1。

抛出异常

PDF电子书基地 http://dayo1982.400gb.com

IOException：如果由于文件结束之外的某种原因不能读取第一个字节，或者随机访问文件已关闭，或者发生其他 I/O 错误。

NullPointerException：如果 b 为 null。

IndexOutOfBoundsException：如果 off 为负，len 为负，或者 len 大于 b.length off。

示例

本示例按读写方式打开一个随机访问文件流，并通过 read()方法读取了 9 个数据字节放到从数组的下标 1 开始放到数组 bs 中并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte[] bs = new byte[10];
    r.read(bs , 1 , 9);
    for (int i = 0; i < bs.length; i++) {
        System.out.println(bs[i]);
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.8 readBoolean 方法——从文件读取一个 boolean 值

从此文件读取一个 boolean。此方法从该文件的当前文件指针开始读取单个字节。值 0 表示 false，其他任何值表示 true。在读取了该字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final boolean readBoolean() throws IOException
```

返回值：读取的 boolean 值。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readBoolean()方法从此文件读取一个 boolean 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r.readBoolean());
```

```
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.16.9 readByte 方法——从文件读取一个 byte 值

从此文件读取一个有符号的八位值。此方法从该文件的当前文件指针开始读取一个字节。如果读取的字节为 b，其中 0≤b≤255，则结果将是(byte)(b)，在读取了该字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final byte readByte() throws IOException
```

返回值：以有符号的 8 位 byte 形式返回此文件的下一个字节。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readByte()方法从此文件读取一个 byte 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r. readByte ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.10 readInt 方法——从文件读取一个 int 值

从文件中读取一个 int 值（4 个字节）。该方法从该文件的当前文件指针开始读取 4 个字节。

方法声明

```
public final int readInt() throws IOException
```

抛出异常：IOException。

示例

本示例通过 RandomAccessFile 类，从指定文件将通过 writeInt 方法写入的 10 个数字，输出在控制台上。

```
File file = new File("H://Getif.java");
RandomAccessFile raf = new RandomAccessFile(file, "rw");
```


PDF电子书基地 http://dayo1982.400gb.com

```
for (int j = 10; j >= 0; j--) {
    raf.seek(j * 4);                          //每隔 4 字节往前读取一个整数
    System.out.println(raf.readInt());
}
raf.close();
```

3.16.11 readChar 方法——从文件读取一个 char 值

从文件中读取一个字符。此方法从该文件的当前文件指针开始读取两个字节。

方法声明

```
public final char readChar() throws IOException
```

返回值：此文件的下两个字节，解释为 char。

抛出异常：IOException。

示例

本示例创建 RandomAccessFile 类对象，通过 readChars()方法将文件中的字符输出。

```
File file = new File("H://Getif.java");
RandomAccessFile raf = new RandomAccessFile(file, "rw");
for(int i =0 ;i<str.length();i++){
    raf.seek(i*2);                            //每隔 2 字节读取一个字符
    System.out.println(raf.readChar());
}
raf.close();
```

3.16.12 readDouble 方法——从文件读取一个 double 值

从此文件读取一个 double。此方法从当前文件指针开始读取一个 long 值，类似于使用 readLong 方法，然后使用 Double 类中的 longBitsToDouble 方法将该 long 转换为一个 double。在读取了这 8 字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final double readDouble() throws IOException
```

返回值：此文件的下 8 字节，解释为一个 double。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readDouble()方法从此文件读取一个 double 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r. readDouble ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.13 readFloat 方法——从文件读取一个 float 值

从此文件读取一个 float。此方法从当前文件指针开始读取一个 int 值，类似于使用 readInt 方法，然后使用 Float 类中的 intBitsToFloat 方法将该 int 转换为一个 float。在读取了这 8 字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final float readFloat() throws IOException
```

返回值：此文件的下 4 字节，解释为一个 float。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readFloat()方法从此文件读取一个 float 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r. readFloat ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.14 readFully 方法——从文件读取一个字节数组

将 b.length 个字节从此文件读入 byte 数组，并从当前文件指针开始。在读取到请求数量的字节之前，此方法将从该文件重复读取。在读取了请求数量的字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明 1

```
public final void readFully(byte[] b) throws IOException
```

入口参数：b 为将数据读入的缓冲区。

PDF电子书基地 http://dayo1982.400gb.com

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readFully()方法将 b.length 个字节从此文件读入 byte 数组中并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte[] bs = new byte[10];
    r.readFully(bs);
    for (int i = 0; i < bs.length; i++) {
        System.out.println(bs[i]);
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

将正好 len 个字节从此文件读入 byte 数组，并从当前文件指针开始。在读取到请求数量的字节之前，此方法将从该文件重复读取。在读取了请求数量的字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

```
public final void readFully(byte[] b, int off, int len) throws IOException
```

入口参数

b：读入数据的缓冲区。

off：写入数据的数组 b 中的初始偏移量。

len：读取的最多字节数。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readFully()方法读取了 9 个数据字节放到从数组的下标 1 开始放到数组 bs 中并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
```

```
        byte[] bs = new byte[10];
        r.readFully(bs, 1, 9);
        for (int i = 0; i < bs.length; i++) {
            System.out.println(bs[i]);
        }
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.16.15 readLine 方法——从此文件读取文本的下一行

从此文件读取文本的下一行。此方法可以从该文件的当前文件指针处成功地读取字节，直到到达行结束符或文件的末尾。每个字节都转换为一个字符，方法是采用该字符的低 8 位字节值，并将该字符的高 8 位设置为 0。因此，此方法不支持完整的 Unicode 字符集。文本行由回车符('\r')和一个换行符('\n')结束，回车符后面紧跟一个换行符，或者是文件的末尾。不使用行结束符，并且在返回的字符串中不包括结束符。在读取了一个换行符、读取了一个回车符和它后面的字节（查看是否为一个新行），到达文件的末尾或者抛出异常之前，此方法一直阻塞。

方法声明

```
public final String readLine() throws IOException
```

返回值：此文件文本的下一行，如果连一个字节也没有读取就已到达文件的末尾，则返回 null。

抛出异常

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readLine()方法从此文件读取一行数据并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    String line = r.readLine();
    System.out.println(line);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.16 readLong 方法——从文件读取一个 long 值

从此文件读取一个有符号的 64 位整数。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
public final long readLong() throws IOException
```

返回值：此文件的下 8 字节，解释为一个 long。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readDouble()方法从此文件读取一个 long 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r.readLong ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.17 readShort 方法——从文件读取一个 short 值

从此文件读取一个有符号的 16 位数。

方法声明

```
public final short readShort() throws IOException
```

返回值：此文件的下 8 字节，解释为一个 long。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readShort()方法从此文件读取一个 short 并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r.readShort ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.18 readUnsignedByte 方法——从文件读取无符号 byte 值

从此文件读取一个无符号的 8 位数。此方法从此文件的当前文件指针开始读取一个字节，并返回该字节。在读取了该字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final int readUnsignedByte() throws IOException
```

返回值：此文件的下一个字节，解释为一个无符号的 8 位数。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readUnsignedByte()方法从此文件读取一个无符号的 8 位数并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r. readUnsignedByte ());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.19 readUnsignedShort 方法——从文件读取无符号 short 值

从此文件读取一个无符号的 16 位数。

方法声明

```
public final int readUnsignedShort() throws IOException
```

返回值：此文件的下一个字节，解释为一个无符号的 8 位数。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readUnsignedShort()方法从此文件读取一个无符号的 16 位数并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
```


PDF电子书基地 http://dayo1982.400gb.com

```
        System.out.println(r. readUnsignedShort ());
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.16.20 readUTF 方法——从此文件读取一个 UTF-8 字符串

从此文件读取一个字符串。该字符串已使用 UTF-8 修改版格式进行编码。

从当前文件指针开始读取前两个字节，类似于使用 readUnsignedShort。此值给出已编码字符串中随后的字节数，而不是结果字符串的长度。随后的字节然后解释为 UTF-8 修改版格式的字节编码字符，并转换为字符。

在读取了所有字节、检测到流的末尾或者抛出异常前，此方法一直阻塞。

方法声明

```
public final String readUTF() throws IOException
```

返回值：一个 Unicode 字符串。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 readUTF()方法从此文件读取一个 Unicode 字符串并输出，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    System.out.println(r.readUTF());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.21 setLength 方法——设置此文件的长度

设置此文件的长度。如果 length 方法返回的文件的现有长度大于 newLength 参数，则该文件将被截短。在此情况下，如果 getFilePointer 方法返回的文件偏移量大于 newLength，那么在返回此方法后，该偏移量将等于 newLength。

如果 length 方法返回的文件的现有长度小于 newLength 参数，则该文件将被扩展。在此情况下，未定义文件扩展部分的内容。

方法声明

```
public void setLength(long newLength) throws IOException
```

入口参数：newLength 为文件的所需长度。

抛出异常

EOFException：如果此文件已到达末尾。

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 setLength()方法设置了文件的长度为 10，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.setLength(10);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.22 seek 方法——将文件指针移至指定位置

该方法将文件指针移至指定位置处。

方法声明

```
public void seek(long lon) throws IOException
```

入口参数：lon 是从文件开头以字节开头以字节为单位测量偏移量位置，在该位置设置文件指针。lon 以字节为单位。

示例

本示例通过 RandomAccessFile 类向指定文件的指定位置中写入数据信息。

```
File file = new File("H://Getif.java");
try {
    RandomAccessFile raf = new RandomAccessFile(file,"rw");
    String str = "asd";
    raf.seek(100);
    raf.writeChars(str);
} catch (Exception e) {
    e.printStackTrace();
}
```

3.16.23 skipBytes 方法——跳过并丢弃 n 个字节

尝试跳过输入的 *n* 个字节以丢弃跳过的字节。此方法可能跳过一些较少数量的字节（可能包括 0）。这可能由任意数量的条件引起；在跳过 *n* 个字节之前已到达文件的末尾只是其中的一种可能。此方法从不抛出 EOFException。返回跳

PDF电子书基地 http://dayo1982.400gb.com

过的实际字节数。如果 *n* 为负数，则不跳过任何字节。

方法声明

```
public int skipBytes(int n) throws IOException
```

入口参数：*n* 为要跳过的字节数。

返回值：跳过的实际字节数。

抛出异常

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 skipBytes()方法跳过了 3 个字节然后输出一个字节，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.skipBytes(3);
    System.out.println(r.read());
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.24 write 方法——向此文件写入指定的字节

向此文件写入指定的字节。从当前文件指针开始写入。

方法声明 1

```
public void write(int b) throws IOException
```

入口参数：b 为要写入的 byte。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 write()方法在文件末尾写入一个字节，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte b = 65;
    r.seek(r.length());
    r.write(65);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 2

将 b.length 个字节从指定 byte 数组写入到此文件，并从当前文件指针开始。

```
public void write(byte[] b) throws IOException
```

入口参数：b 为数据。

抛出异常

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 write()方法在文件末尾向文件中写入了数组了中的值，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte[] bs = {65, 66, 67, 68, 69, 70};
    r.seek(r.length());
    r.write(bs);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 3

将 len 个字节从指定 byte 数组写入到此文件，并从偏移量 off 处开始。

```
public void write(byte[] b, int off, int len) throws IOException
```

入口参数

b：数据。

off：数据的初始偏移量。

len：要写入的字节数。

抛出异常

IOException：如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 write()方法在文件末尾向文件写入了以数组下标为 1～3 的值，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    byte[] bs = {65, 66, 67, 68, 69, 70};
    r.seek(r.length());
    r.write(bs, 1, 3);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.16.25 writeBoolean 方法——将 boolean 写入文件

按单字节值将 boolean 写入该文件。值 true 写出为值(byte)1；值 false 写出为值(byte)0。写入从文件指针的当前位置开始。

方法声明

```
public final void writeBoolean(boolean v) throws IOException
```

入口参数：v 为要写入的 boolean 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeBoolean()方法在文件末尾写入一个 boolean，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    boolean v = true;
    r.writeBoolean(v);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.26 writeByte 方法——将 byte 写入该文件

按单字节值将 byte 写入该文件。写入从文件指针的当前位置开始。

方法声明

```
public final void writeByte(int v) throws IOException
```

入口参数：v 为要写入的 byte 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeByte()方法在文件末尾写入一个 byte，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    byte v = 65;
    r. writeByte (v);
    r.close();
} catch (IOException e) {
```

```
        e.printStackTrace();
    }
```

3.16.27 writeBytes 方法——按字节序列将该字符串写入该文件

按字节序列将该字符串写入该文件。该字符串中的每个字符均按顺序写出，并丢弃其高 8 位。写入从文件指针的当前位置开始。

方法声明

```
public final void writeBytes(String s) throws IOException
```

入口参数：s 为要写入的字节的字符串。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeBytes()方法在文件末尾写入一个字符串，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    String s = "i like java";
    r.writeBytes(s);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.28 writeChars 方法——写入作为字符数据的字符串

向文件写入一个作为字符数据的字符串。

方法声明

```
public final void writeChars(String str) throws IOException
```

入口参数：str 是要写入流的字符串。

抛出异常：IOException。

示例

本示例通过 RandomAccessFile 类，通过 writeChars()方法向指定文件中写入字符串。

```
File file = new File("H://Getif.java");
RandomAccessFile raf = new RandomAccessFile(file, "rw");
String str = "Java 编程词典";
raf.writeChars(str);
raf.close();
```


PDF电子书基地 http://dayo1982.400gb.com

3.16.29 writeDouble 方法——写入 double 值

使用 Double 类中的 doubleToLongBits 方法将双精度参数转换为一个 long，然后按 8 字节数量将该 long 值写入该文件，先定高字节。写入从文件指针的当前位置开始。

方法声明

```
public final void writeDouble(double v) throws IOException
```

入口参数：v 为要写入的 double 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeDouble()方法在文件末尾写入一个 double，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    double v = 123.456d;
    r.writeDouble(v);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.30 writeFloat 方法——向文件写入 float 值

使用 Float 类中的 floatToIntBits 方法将浮点参数转换为一个 int，然后按 4 字节数量将该 int 值写入该文件，先写高字节。写入从文件指针的当前位置开始。

方法声明

```
public final void writeFloat(float v) throws IOException
```

入口参数：v 为要写入的 float 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeFloat()方法在文件末尾写入一个 float，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    float v = 123.456f;
```

```
        r. writeFloat(v);
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.16.31 writeInt 方法——向文件写入 int 值

向文件写入一个 int 值。

方法声明

```
public final void writeInt(int i) throws IOException
```

入口参数：I 是要写入的 int 值。

抛出异常：IOException。

示例

本示例通过 RandomAccessFile 类把几个 int 型整数，写入一个指定文件中。

```
File file = new File("H://Getif.java");
    RandomAccessFile raf = new RandomAccessFile(file, "rw");
    int date[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    for (int i = 0; i < date.length; i++) {
        raf.writeInt(date[i]);
    }
```

3.16.32 writeLong 方法——向文件写入 long 值

按 8 个字节将 long 写入该文件，先写高字节。写入从文件指针的当前位置开始。

方法声明

```
public final void writeLong(long v) throws IOException
```

入口参数：v 为要写入的 long 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeLong()方法在文件末尾写入一个 long，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    long v = 123;
    r. writeLong(v);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

3.16.33 writeShort 方法——向文件写入 short 值

按两个字节将 short 写入该文件，先写高字节。写入从文件指针的当前位置开始。

方法声明

```
public final void writeShort(int v) throws IOException
```

入口参数：v 为要写入的 short 值。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeShort()方法在文件末尾写入一个 short，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
    r.seek(r.length());
    short v = 12;
    r. writeShort(v);
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.16.34 writeUTF 方法——向文件写入 UTF-8 编码的字符串

使用 modified UTF-8 编码以与机器无关的方式将一个字符串写入该文件。

首先，把两个字节从文件的当前文件指针写入此文件，类似于使用 writeShort 方法并给定要跟随的字节数。此值是实际写出的字节数，而不是该字符串的长度。在该长度之后，按顺序输出该字符串的每个字符，并对每个字符使用 UTF-8 修改版编码。

方法声明

```
public final void writeUTF(String str) throws IOException
```

入口参数：v 为要写入的字符串。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例按读写方式打开一个随机访问文件流，并通过 writeUTF()方法在文件末尾写入一个字符串，最后调用 close()方法将流关闭。

```
try {
    File file = new File("H://Getif.java");
    RandomAccessFile r = new RandomAccessFile(file, "rw");
```

```
        r.seek(r.length());
        String v = "编程词典";
        r.writeUTF(v);
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.17 Reader 类——读取字符流的抽象类

java.io 包中的 Reader 类是读取字符流的抽象类。字符流是以字符为单位进行输入输出，该类定义了操作字符输入流的各种方法。

3.17.1 close 方法——关闭流

关闭该流并释放与之关联的所有资源。

方法声明

```
public abstract void close() throws IOException
```

抛出异常：IOException。

示例

本示例创建 Reader 子类 InputStreamReader 实例，并将其初始化为控制台的字符输入流，从字符输入流中获取一些字符信息到字符数组中，用来给字符数组创建一个字符串，并输出到控制台中，最后通过 close()方法将流关闭。

```
InputStreamReader rin=new InputStreamReader(System.in);
try {
    char[] cs=new char[50];
    rin.read(cs);
    String str=new String(cs);
    System.out.println("输入的字符串内容：\n\t\t"+str.trim());
    rin.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.17.2 mark 方法——标记流中的当前位置

标记流中的当前位置，对 reset()的后续调用将尝试将该流重新定位到此点，并不是所有的字符输入流都支持 mark()操作。

方法声明

```
public void mark(int readAheadLimit) throws IOException
```

入口参数：readAheadLimit 为在仍保留该标记的情况下，对可读取字符数量的限制。在读取这样多的字符后，尝试重置流可能会失败。

抛出异常：IOException，如果该流不支持 mark()，或者发生一些其他 I/O 错误。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

示例

本示例创建字符输入流对象，实现循环读取指定磁盘文件中前 5 个字符。

```
try {
    File file = new File("H://Getif.java"); // 用文件路径新建一个文件对象
    //创建输入流
    Reader r = new BufferedReader(new FileReader(file));
    int count = 0;
    r.mark(50);
    for (int i = 0; i < file.length(); i++) {
        int read = r.read();
        if (count % 5 == 0) {
            r.reset();
            System.out.println();
        }
        count++;
        System.out.print((char) read);
    }
    r.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.17.3 markSupported 方法——判断是否支持 mark() 操作

判断此流是否支持 mark()操作，默认实现始终返回 false，子类应重写此方法。

方法声明

```
public boolean markSupported()
```

返回值：当且仅当此流支持此 mark 操作时，返回 true。

示例

本示例创建字符输入流对象，并通过 markSupported()方法输出了是否支持 mark()操作，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");     // 用文件路径新建一个文件对象
try {
    Reader r = new FileReader(file);
    if(r.markSupported()){
        System.out.println("支持 mark()操作");
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.17.4 read 方法——读取流中的数据

read()方法用于读取流中的数据，该方法提供了多种重载形式。

方法声明 1

```
public int read(char[] cbuf) throws IOException
```

返回值：返回读取的字符数，如果已到达流的末尾，则返回-1。

抛出异常：IOException。

示例

本示例创建 Reader 子类 InputStreamReader 实例，并将其初始化为控制台的字符输入流，创建字符数组，通过调用 read()方法读取流中的数据，最后将流关闭。

```
InputStreamReader rin=new InputStreamReader(System.in);
try {
    char[] cs=new char[100];
    rin.read(cs);
    String str=new String(cs);
    System.out.println("输入的字符串内容：\n"+str.trim());
    rin.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

运行上述代码，在控制台上输入“你好”，控制台上将输出“"输入的字符串内容：你好”，如下图所示。



字符输入流

方法声明 2

```
public int read(char[] cbuf,int off,int len) throws IOException
```

从流中读入数组的某一部分。

入口参数

cbuf：字符数组。

off：开始存储字符数组的下标。

len：要读取字符的最多字符数。

返回值：返回读取的字符数，如果已到达流的末尾，则返回-1。

抛出异常：IOException。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例创建 Reader 子类 InputStreamReader 实例，并将其初始化为控制台的字符输入流，实现从流中读取指定长度的字符，在控制台上输出。

```
InputStreamReader rin=new InputStreamReader(System.in);
try {
    char[] cs=new char[100];
    rin.read(cs,0,10);
    String str=new String(cs);
    System.out.println("输入的字符串内容：\n"+str.trim());
    rin.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

运行程序，实现向控制台中输入字符串，而控制台上输出的是长度为 10 的字符，运行结果如下图所示。



字符输入流

3.17.5 ready 方法——判断是否准备读取此流

判断是否准备读取此流。

方法声明

```
public boolean ready() throws IOException
```

返回值：如果保证下一个 read()不阻塞输入，则返回 true，否则返回 false。注意，返回 false 并不保证阻塞下一次读取。

抛出异常：IOException，如果发生 I/O 错误。

示例

本示例创建字符输入流对象，并通过 ready()方法判断是否准备读取此流，如果为 true 则开始读取字符，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    Reader r = new FileReader(file);
    if(r.ready()){    //判断是否准备读取此流
        int i;
        while((i = r.read()) != -1){      //开始读取
```

```
                System.out.println((char)i);        //输出字符
            }
        }
        r.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

3.17.6 reset 方法——重置该流

重置该流。

方法声明

```
public void reset() throws IOException
```

抛出异常：IOException，如果尚未标记流，或者该标记已失效，或者该流不支持 reset()，或者发生一些其他 I/O 错误。

示例

本示例创建字符输入流对象，实现循环读取指定磁盘文件中前 5 个字符。

```
try {
    File file = new File("H://Getif.java"); // 用文件路径新建一个文件对象
    //创建输入流
    Reader r = new BufferedReader(new FileReader(file));
    int count = 0;
    r.mark(50);
    for (int i = 0; i < file.length(); i++) {
        int read = r.read();
        if (count % 5 == 0) {
            r.reset();
            System.out.println();
        }
        count++;
        System.out.print((char) read);
    }
    r.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

3.17.7 skip 方法——跳过并丢弃 *n* 个字节

从输入流中跳过并丢弃 *n* 个字节的数据。

方法声明

```
public long skip(long n) throws IOException
```

返回值：实际跳过的字节数。

抛出异常：IOException，如果 *n* 为负，如果该流不支持查找操作，或者发生

PDF电子书基地 http://dayo1982.400gb.com

I/O 错误。

说明

出于各种原因，skip 方法结束时跳过的字节数可能小于该数，也可能为 0。导致这种情况的原因很多，跳过 *n* 个字节之前已到达文件末尾只是其中一种可能。返回跳过的实际字节数。如果 *n* 为负，则不跳过任何字节。此类的 skip 方法创建一个 byte 数组，然后重复将字节读入其中，直到读够 *n* 个字节或已到达流末尾为止。建议子类提供此方法更为有效的实现。例如，可依赖搜索能力的实现。

示例

本示例创建字节输入流对象，并通过 skip()方法跳过 10 个字节然后开始读数据并输出，最后调用 close()方法将流关闭。

```
File file = new File("H:\\Getif.java");      // 用文件路径新建一个文件对象
try {
    Reader r = new FileReader(file);
    if(r.ready()){    //判断是否准备读取此流
        r.skip(10); //跳过 10 个字符
        int i;
        while((i = r.read()) != -1){      //开始读取
            System.out.println((char)i);      //输出字符
        }
    }
    r.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.18 Scanner 类——简单文本扫描器

java.util 包中的 Scanner 类为一个可以使用正则表达式来解析基本类型和字符串的简单文本扫描器。

3.18.1 构造方法——Scanner 类的构造方法

Scanner 类常用的构造方法如下表所示。

Scanner 类的构造方法

构 造 方 法	使 用 说 明
Scanner(File file)	创建一个从指定文件扫描的 Scanner 对象
Scanner(InputStream inp)	创建一个从指定输入流扫描的 Scanner 对象
Scanner(String str)	创建一个从指定字符串扫描的 Scanner 对象

3.18.2 close 方法——关闭扫描器

关闭扫描器。

方法声明

```
public void close()
```

示例

本示例创建了一个扫描器，调用 close()方法将其关闭。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.close();
```

3.18.3 delimiter 方法——返回匹配分隔符的 Pattern

返回 Scanner 当前正在用于匹配分隔符的 Pattern。

方法声明

```
public Pattern delimiter()
```

返回值：此扫描器的分隔模式。

示例

本示例创建了一个扫描器，调用 delimiter()方法输出了此扫描器的分隔模式。

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\sfish\\s");
System.out.println(s.delimiter());
s.close();
```

3.18.4 findInLine 方法——忽略分隔符查找下一个构造模式

该方法在忽略分隔符的情况下查找下一个从指定字符串构造的模式。

方法声明 1

```
public String findInLine(String str)
```

入口参数：str 指定要搜索的模式的字符串。

返回值：与指定模式匹配的字符串。

示例

本示例使用正则表达式解析字符，并将满足条件的字符输出。

```
String input = "1 and 2 and yellow and blue and";
Scanner s = new Scanner(input);
s.findInLine("(\\d+) and (\\d+) and (\\w+) and (\\w+)");
MatchResult result = s.match();
for (int i=1; i<=result.groupCount(); i++)
    System.out.println(result.group(i));
s.close();
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

该方法在忽略分隔符的情况下查找下一个从指定字符串构造的模式。

```
public String findInLine(Pattern pattern)
```

入口参数：pattern 为要扫描的模式。

返回值：与指定模式匹配的文本。

抛出异常：IllegalStateException，如果此扫描器已关闭。

注意

如果在下一个行分隔符前找到了该模式，则扫描器执行匹配的输入信息，并返回与该模式匹配的字符串。如果未找到，则返回 null，且扫描器的位置不变。在等待与该模式匹配的输入信息时，此方法可能阻塞。

因为此方法会继续在输入信息中查找指定的模式，所以如果没有行分隔符，它可能会缓冲所有输入信息，并查找所需的标记。

示例

本示例使用扫描模式解析字符，并将满足条件的字符输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.findInLine(Pattern.compile("(\\d+) and (\\d+) *"));
MatchResult result = s.match();
for (int i = 1; i <= result.groupCount(); i++) {
    System.out.println(result.group(i));
}
s.close();
```

3.18.5 findWithinHorizon 方法——试图查找下一个指定模式

试图查找下一个指定模式。

方法声明 1

```
public String findWithinHorizon(Pattern pattern, int horizon)
```

入口参数：pattern 为要扫描的模式。

返回值：与指定模式匹配的文本。

抛出异常

IllegalStateException：如果此扫描器已关闭。

IllegalArgumentException：如果 horizon 为负。

注意

此方法在忽略分隔符的情况下在指定的搜索范围内搜索输入信息。如果找到了该模式，则扫描器执行匹配的输入信息，并返回与该模式匹配的字符串。如果未找到，则返回 null，且扫描器的位置不变。在等待与该模式匹配的输入信息时，

此方法可能阻塞。扫描器的搜索永远都不超出其当前位置以外的 horizon 代码点。注意，匹配可能被 horizon 剪裁；也就是说，如果 horizon 变大，则任意匹配结果都可能不同。扫描器将 horizon 看作一个不固定的透明边界（请参见 Matcher.useTransparentBounds(boolean)和 Matcher.useAnchoringBounds(boolean)）。如果 horizon 为 0，则忽略 horizon，此方法继续在输入信息中查找指定的模式，且不设边界。在这种情况下，它可能会缓冲所有输入信息，并搜索该模式。

示例

本示例创建了一个扫描器，调用 findWithinHorizon()方法输出了与指定模式匹配的文本。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
String str = s.findWithinHorizon(Pattern.compile("\\sand\\s\\d"),7);
System.out.println(str);
s.close();
```

方法声明 2

试图在忽略分隔符的情况下查找下一个从指定字符串构造的模式。

```
public String findWithinHorizon(String pattern, int horizon)
```

入口参数：pattern 为要扫描的模式。

返回值：与指定模式匹配的文本。

抛出异常

IllegalStateException：如果此扫描器已关闭。

IllegalArgumentException：如果 horizon 为负。

示例

本示例创建了一个扫描器，调用 findWithinHorizon()方法输出了与指定模式匹配的文本。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
String str = s.findWithinHorizon(Pattern.compile("\\sand\\s\\d"),7);
System.out.println(str);
s.close();
```

3.18.6 hasNext 方法——扫描器的输入中是否有另一个标记

如果此扫描器的输入中有另一个标记，则返回 true。在等待要扫描的输入时，此方法可能阻塞。扫描器将不执行任何输入。

方法声明 1

```
public boolean hasNext()
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：当且仅当此扫描器有另一个标记时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNext()方法输出了扫描器是否有另一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext()){
    System.out.println("扫描器有另一个标记");
}
s.close();
```

方法声明 2

如果下一个完整标记与指定模式匹配，则返回 true。完整标记以与分隔模式匹配的输入信息为前后缀。在等待输入信息时，此方法可能阻塞。扫描器不执行任何输入。

```
public boolean hasNext(Pattern pattern)
```

返回值：当且仅当此扫描器有另一个标记与指定模式匹配时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNext()方法输出了扫描器在指定模式下是否有另一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext(Pattern.compile("\\d"))){
    System.out.println("扫描器有另一个标记");
}
s.close();
```

方法声明 3

如果下一个标记与从指定字符串构造的模式匹配，则返回 true，扫描器不执行任何输入。

```
public boolean hasNext(String pattern)
```

返回值：当且仅当此扫描器有另一个标记与指定模式匹配时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNext()方法输出了扫描器在指定模式下是否有另一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext("\\d")){
    System.out.println("扫描器有另一个标记");
}
s.close();
```

3.18.7 hasNextBigDecimal 方法——下一个标记是否默认基数中的一个 BigDecimal

如果通过使用 nextBigDecimal()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 BigDecimal，则返回 true，扫描器不执行任何输入。

方法声明

```
public boolean hasNextBigDecimal()
```

返回值：当且仅当此扫描器的下一个标记是有效的 BigDecimal 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigDecimal()判断下一个标记是否为有效的 BigDecimal 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBigDecimal()){
    System.out.println(s.nextBigDecimal());
}
s.close();
```

3.18.8 hasNextBigInteger 方法——下一个标记是否默认基数中的一个 BigInteger

如果通过使用 nextBigInteger()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 BigInteger 值，则返回 true。扫描器不执行任何输入。

方法声明 1

```
public boolean hasNextBigInteger()
```

返回值：当且仅当此扫描器的下一个标记是有效的 BigInteger 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigInteger()判断下一个标记是否为有效的 BigInteger 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
```


PDF电子书基地 http://dayo1982.400gb.com

```
if(s.hasNextBigInteger()){
    System.out.println(s.nextBigInteger());
}
s.close();
```

方法声明 2

如果通过使用 nextBigInteger()方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 BigInteger 值，则返回 true。扫描器不执行任何输入。

```
public boolean hasNextBigInteger(int radix)
```

入口参数：radix 为用于将标记解释为一个 integer 的基数。

返回值：当且仅当此扫描器的下一个标记是有效的 BigInteger 时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigInteger()判断下一个标记是否为有效基数 10 的 BigInteger 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBigInteger(10)){
    System.out.println(s.nextBigInteger(10));
}
s.close();
```

3.18.9 hasNextBoolean 方法——下一个标记是否为布尔值

如果通过使用一个从字符串“true|false”创建的大小写敏感的模式，此扫描器输入信息中的下一个标记可以解释为一个布尔值，则返回 true。扫描器不执行匹配的输入。

方法声明

```
public boolean hasNextBoolean()
```

返回值：当且仅当此扫描器的下一个标记是有效的布尔值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBoolean()判断下一个标记是否为有效的 boolean 值，如果有效则输出。

```
String input = "true and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBoolean()){
    System.out.println(s.nextBoolean());
}
s.close();
```

3.18.10 hasNextByte 方法——下一个标记是否为 byte 值

如果通过使用 nextByte()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个字节值，则返回 true。扫描器不执行任何输入。

方法声明 1

```
public boolean hasNextByte()
```

返回值：当且仅当此扫描器的下一个标记是有效的字节值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigInteger()判断下一个标记是否为有效的 byte 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextByte()){
    System.out.println(s.nextByte());
}
s.close();
```

方法声明 2

如果通过使用 nextByte()方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个字节值，则返回 true。扫描器不执行任何输入。

```
public boolean hasNextByte(int radix)
```

入口参数：radix 用于将标记解释为一个字节值的基数。

返回值：当且仅当此扫描器的下一个标记是有效的字节值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigInteger()判断下一个标记是否为有效的 10 进制 byte 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextByte(10)){
    System.out.println(s.nextByte(10));
}
s.close();
```

3.18.11 hasNextDouble 方法——下一个标记是否为 double 值

如果通过使用 nextDouble()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 double 值，则返回 true。扫描器不执行任何输入。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
public boolean hasNextDouble()
```

返回值：当且仅当此扫描器的下一个标记是有效的 double 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextDouble()判断下一个标记是否为有效的 double 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextDouble()){
    System.out.println(s.nextDouble());
}
s.close();
```

3.18.12 hasNextFloat 方法——下一个标记是否为 float 值

如果通过使用 nextFloat()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 float 值，则返回 true。扫描器不执行任何输入。

方法声明

```
public boolean hasNextFloat()
```

返回值：当且仅当此扫描器的下一个标记是有效的 double 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextFloat()判断下一个标记是否为有效的 float 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextFloat ()){
    System.out.println(s.nextFloat ());
}
s.close();
```

3.18.13 hasNextInt 方法——下一个标记是否为 int 值

如果通过使用 nextInt()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 int 值，则返回 true。扫描器不执行任何输入。

方法声明 1

```
public boolean hasNextInt()
```

返回值：当且仅当此扫描器的下一个标记是有效的 int 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextInt()判断下一个标记是否为有效的 int 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextInt()){
    System.out.println(s.nextInt());
}
s.close();
```

方法声明 2

如果通过使用 nextInt()方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 int 值，则返回 true。扫描器不执行任何输入。

```
public boolean hasNextInt(int radix)
```

入口参数：radix 用于将标记解释为一个 int 值的基数。

返回值：当且仅当此扫描器的下一个标记是有效的 int 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextBigInteger()判断下一个标记是否为有效的十进制 int 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextInt(10)){
    System.out.println(s.nextInt(10));
}
s.close();
```

3.18.14 hasNextLine 方法——是否存在下一行

如果在此扫描器的输入中存在另一行，则返回 true。在等待输入信息时，此方法可能阻塞。扫描器不执行任何输入。

方法声明

```
public boolean hasNextLine()
```

返回值：当且仅当此扫描器有另一行输入时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextLine()判断此扫描器的输入中是否存在另一行，如果有效则输出。

PDF电子书基地 http://dayo1982.400gb.com

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLine()){
    System.out.println(s.nextLine());
}
s.close();
```

3.18.15 hasNextLong 方法——下一个标记是否为 long 值

如果通过使用 nextLong()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 long 值，则返回 true。扫描器不执行任何输入。

方法声明 1

```
public boolean hasNextLong()
```

返回值：当且仅当此扫描器的下一个标记是有效的 long 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextLong()判断下一个标记是否为有效的 long 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLong()){
    System.out.println(s.nextLong());
}
s.close();
```

方法声明 2

如果通过使用 nextLong()方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 long 值，则返回 true。扫描器不执行任何输入。

```
public boolean hasNextLong(int radix)
```

入口参数：radix 用于将标记解释为一个 long 值的基数。

返回值：当且仅当此扫描器的下一个标记是有效的 long 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextLong()判断下一个标记是否为有效的 10 进制 long 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLong(10)){
    System.out.println(s.nextLong(10));
}
s.close();
```

3.18.16 hasNextShort 方法——下一个标记是否为 short 值

如果通过使用 nextShort()方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 short 值，则返回 true。扫描器不执行任何输入。

方法声明 1

```
public boolean hasNextShort()
```

返回值：当且仅当此扫描器的下一个标记是默认基数中的有效的 short 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextShort()判断下一个标记是否为有效的 short 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextShort()){
    System.out.println(s.nextShort());
}
s.close();
```

方法声明 2

如果通过使用 nextShort()方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 short 值，则返回 true。扫描器不执行任何输入。

```
public boolean hasNextShort(int radix)
```

入口参数：radix 用于将标记解释为一个 short 值的基数。

返回值：当且仅当此扫描器的下一个标记是有效的 short 值时才返回 true。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 hasNextShort()判断下一个标记是否为有效的十进制 short 值，如果有效则输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextShort (10)){
    System.out.println(s. nextShort (10));
}
s.close();
```

3.18.17 ioException 方法——最后抛出的 IOException

返回此 Scanner 的底层 Readable 最后抛出的 IOException。如果不存在这样的异常，则此方法返回 null。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
public IOException ioException()
```

返回值：此扫描器的 readable 最后抛出的异常。

示例

本示例创建一个扫描器，判断是否存在 IOException，如果有效则输出错误信息。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
IOException ie = s.ioException();
if(ie != null){
    ie.printStackTrace();
}
s.close();
```

3.18.18 locale 方法——返回此扫描器的语言环境

返回此扫描器的语言环境。

方法声明

```
public Locale locale()
```

返回值：此扫描器的语言环境。

示例

本示例创建一个扫描器，通过 locale()方法输出了此扫描器的语言环境。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
System.out.println(s.locale());
s.close();
```

3.18.19 match 方法——最后扫描操作的匹配结果

返回此扫描器所执行的最后扫描操作的匹配结果。如果未执行匹配操作，或者最后的匹配操作失败，则此方法抛出 IllegalStateException。

方法声明

```
public MatchResult match()
```

返回值：最后一次匹配操作的匹配结果。

注意

如果 Scanner 的不同 next 方法执行结束时未抛出异常，则它们会给出一个匹配结果。例如，调用了 nextInt()方法（它返回一个 int）后，此方法返回一个 MatchResult，作为搜索上述 Integer 正则表达式的结果。类似地，如果 findInLine(java.lang.String)、findWithinHorizon(java.lang.String,int)和 skip(java.util.regex.Pattern)方法的调用成

功，则它们会给出一个匹配结果。

示例

本示例创建一个扫描器，通过 match()返回了匹配结果并将其输出。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.findInLine("(\\d+) and (\\d+) and (\\w+) and (\\w+)");
MatchResult result = s.match();
for (int i=1; i<=result.groupCount(); i++)
    System.out.println(result.group(i));
s.close();
```

3.18.20 next 方法——读取下一个完整标记

查找并返回来自此扫描器的下一个完整标记。完整标记的前后是与分隔模式匹配的输入信息。即使以前调用 hasNext()返回了 true，在等待要扫描的输入时此方法也可能阻塞。

方法声明 1

```
public String next()
```

返回值：下一个标记。

抛出异常：IllegalStateException，如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 next()方法输出了扫描器的下一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext()){
    System.out.println(s.next());
}
s.close();
```

方法声明 2

如果下一个标记与指定模式匹配，则返回下一个标记。即使以前调用 hasNext(Pattern)返回了 true，在等待要扫描的输入时此方法也可能阻塞。如果此匹配操作成功，则扫描器执行与该模式匹配的输入。

```
public String next(Pattern pattern)
```

入口参数：pattern 为要扫描的模式。

返回值：下一个标记。

抛出异常

NoSuchElementException：如果没有更多的标记可用。

IllegalStateException：如果此扫描器已关闭。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例创建一个扫描器，通过 next()方法输出了扫描器的下一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext()){
    System.out.println(s.next(Pattern.compile("\\d")));
}
s.close();
```

方法声明 3

如果下一个标记与从指定字符串构造的模式匹配，则返回下一个标记。如果匹配操作成功，则扫描器执行与该模式匹配的输入。此方法调用 next(pattern)的行为与调用 next(Pattern.compile(pattern))完全相同。

```
public String next(String pattern)
```

入口参数：指定要扫描的模式的字符串。

返回值：下一个标记。

抛出异常

NoSuchElementException：如果没有更多的标记可用。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 next()方法输出了扫描器的下一个标记。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNext("\\d")){
    System.out.println(s.next("\\d"));
}
s.close();
```

3.18.21 nextBigDecimal 方法——读取下一个 BigDecimal 对象

将输入信息的下一个标记扫描为一个 BigDecimal。

如果下一个标记与上述的 Decimal 正则表达式匹配，则该标记转换成一个 BigDecimal 值，就好像通过移除所有组分隔符，通过 Character.digit 将非 ASCII 数字映射为 ASCII 数字，以及将得到的字符串传递给 BigDecimal(String)构造方法。

方法声明

```
public BigDecimal nextBigDecimal()
```

返回值：从输入信息扫描的 BigDecimal。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或

者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextBigDecimal()方法输出了扫描到的BigDecimal。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBigDecimal()){
    System.out.println(s.nextBigDecimal());
}
s.close();
```

3.18.22 nextBigInteger 方法——读取下一个 BigInteger 对象

将输入信息的下一个标记扫描为一个 BigInteger。

此方法调用 nextBigInteger()的行为与调用 nextBigInteger(radix)完全相同，其中的 radix 是此扫描器的默认基数。

方法声明 1

```
public BigInteger nextBigInteger()
```

返回值：从输入信息扫描的 BigInteger。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextBigDecimal()方法输出了扫描到的BigInteger。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBigInteger()){
    System.out.println(s.nextBigInteger());
}
s.close();
```

方法声明 2

将输入信息的下一个标记扫描为一个 BigInteger。

如果下一个标记与上述的 Integer 正则表达式匹配，则该标记转换成一个

PDF电子书基地 http://dayo1982.400gb.com

BigInteger 值，就好像通过移除所有组分隔符，通过 Character.digit 将非 ASCII 数字映射为 ASCII 数字，以及将得到的字符串传递给带有指定基数的 BigInteger(String,int)构造方法。

```
public BigInteger nextBigInteger(int radix)
```

入口参数：radix 用于解释标记的基数。

返回值：从输入信息扫描的 BigInteger。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextBigInteger()输出了有效基数为 10 的 BigInteger 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBigInteger(10)){
    System.out.println(s.nextBigInteger(10));
}
s.close();
```

3.18.23 nextBoolean 方法——读取下一个 boolean 值

扫描解释为一个布尔值的输入标记并返回该值。如果下一个标记不能转换为有效的布尔值，则此方法将抛出 InputMismatchException。如果此匹配操作成功，则扫描器执行匹配的输入。

方法声明

```
public boolean nextBoolean()
```

返回值：当且仅当此扫描器的下一个标记是有效的布尔值时才返回 true。

抛出异常

InputMismatchException：如果下一标记不是有效的布尔值。

NoSuchElementException：如果输入信息已耗尽。

示例

本示例创建一个扫描器，通过 nextBoolean()输出了下一个标记的 boolean 值。

```
String input = "true and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextBoolean()){
```

```
        System.out.println(s.nextBoolean());
    }
    s.close();
```

3.18.24 nextByte 方法——读取下一个 byte 值

将输入信息的下一个标记扫描为一个 byte。

此方法调用 nextByte()的行为与调用 nextByte(radix)完全相同，其中的 radix 是此扫描器的默认基数。

方法声明 1

```
public byte nextByte()
```

返回值：从输入信息扫描的 byte。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextByte()输出了下一个标记为有效的 byte 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextByte()){
    System.out.println(s.nextByte());
}
s.close();
```

方法声明 2

将输入信息的下一个标记扫描为一个 byte。如果下一个标记不能转换为有效的字节值，则此方法将抛出 InputMismatchException，如下所述。如果此转换成功，则扫描器执行匹配的输入。

```
public byte nextByte(int radix)
```

入口参数：radix 用于将标记解释为一个字节值的基数。

返回值：从输入信息扫描的 byte。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建一个扫描器，通过 nextByte()输出了下一个标记为有效的十进制 byte 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextByte(10)){
    System.out.println(s.nextByte(10));
}
s.close();
```

3.18.25 nextDouble 方法——读取下一个 double 值

将输入信息的下一个标记扫描为一个 double。如果下一个标记不能转换为有效的 double 值，则此方法将抛出 InputMismatchException。如果此转换成功，则扫描器执行匹配的输入。

方法声明

```
public double nextDouble()
```

返回值：从输入信息扫描的 double。

抛出异常：IllegalStateException 如果此扫描器已关闭。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextDouble()方法转出了下一个标记为有效的 double 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextDouble()){
    System.out.println(s.nextDouble());
}
s.close();
```

3.18.26 nextFloat 方法——读取下一个 float 值

扫描下一个 float 值。如果下一个标记不能转换为有效的 float 值，则此方法将抛出 *InputMismatchException*。

方法声明

```
public float nextFloat()
```

返回值：从输入信息扫描的 float。

示例

本示例通过 Scanner 类对象，将指定文件中 float 值输出。

```
File file = new File("H://Getif.java");
Scanner sca = new Scanner(file);
while(sca.hasNextFloat()){
    System.out.println(sca.nextFloat());
}
```

3.18.27 nextInt 方法——读取下一个 int 值

扫描下一个 int 值。

方法声明 1

```
public int nextInt()
```

返回值：输入信息扫描的 int 值。

示例

本示例通过 Scanner 类对象，将指定文件中 int 值输出。

```
File file = new File("H://Getif.java");
Scanner sca = new Scanner(file);
while(sca.hasNextInt()){
    System.out.println(sca.nextInt());
}
```

方法声明 2

将输入信息的下一个标记扫描为一个 int。如果下一个标记不能转换为有效的 int 值，则此方法将抛出 InputMismatchException，如下所述。如果此转换成功，则扫描器执行匹配的输入。

```
public int nextInt(int radix)
```

入口参数：radix 用于将标记解释为一个 int 值的基数。

返回值：从输入信息扫描的 int。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextInt()方法输出了下一个标记为有效的十进制 int 值。

PDF电子书基地 http://dayo1982.400gb.com

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextInt(10)){
    System.out.println(s.nextInt(10));
}
s.close();
```

3.18.28 nextLine 方法——读取下一行文本

此扫描器执行当前行，并返回跳过的输入信息。此方法返回当前行的其余部分，不包括结尾处的行分隔符。当前位置移至下一行的行首。

因为此方法会继续在输入信息中查找行分隔符，所以如果没有行分隔符，它可能会缓冲所有输入信息，并查找要跳过的行。

方法声明

```
public String nextLine()
```

返回值：跳过的行。

抛出异常

NoSuchElementException：如果未找到这样的行。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextLine()方法输出了此扫描器的输入中存在另一行。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLine()){
    System.out.println(s.nextLine());
}
s.close();
```

3.18.29 nextLong 方法——读取下一个 long 值

将输入信息的下一个标记扫描为一个 long。

此方法调用 nextLong()的行为与调用 nextLong(radix)完全相同，其中的 radix 是此扫描器的默认基数。

方法声明 1

```
public long nextLong()
```

返回值：从输入信息扫描的 long。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextLong()方法输出了下一个标记为有效的 long 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLong()){
    System.out.println(s.nextLong());
}
s.close();
```

方法声明 2

将输入信息的下一个标记扫描为一个 long。如果下一个标记不能转换为有效的 long 值，则此方法将抛出 InputMismatchException，如下所述。如果此转换成功，则扫描器执行匹配的输入。

```
public long nextLong(int radix)
```

入口参数：radix 用于将标记解释为一个 long 值的基数。

返回值：从输入信息扫描的 long。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextLong()方法输出了下一个标记为有效的十进制 long 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextLong(10)){
    System.out.println(s.nextLong(10));
}
s.close();
```

3.18.30 nextShort 方法——读取下一个 short 值

方法声明

将输入信息的下一个标记扫描为一个 short。如果下一个标记不能转换为有效的 short 值，则此方法将抛出 InputMismatchException，如下所述。如果此转换成功，则扫描器执行匹配的输入。

PDF电子书基地 http://dayo1982.400gb.com




```
public short nextShort(int radix)
```

入口参数：radix 用于将标记解释为一个 short 值的基数。

返回值：当且仅当此扫描器的下一个标记是有效的 short 值时才返回 true。

抛出异常

InputMismatchException：如果下一个标记与 Decimal 正则表达式不匹配，或者超出范围。

NoSuchElementException：如果输入信息已耗尽。

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 nextShort()输出了下一个标记为有效的十进制 short 值。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
if(s.hasNextShort (10)){
      System.out.println(s. nextShort (10));
}
s.close();
```

3.18.31 radix 方法——返回扫描器的默认基数

方法声明

返回扫描器的默认基数。

```
public int radix()
```

返回值：扫描器的默认基数。

示例

本示例创建一个扫描器，通过 radix()方法输出了扫描器的默认基数。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
System.out.println(s.radix());
s.close();
```

3.18.32 reset 方法——重置扫描器

方法声明

重置扫描器。

```
public Scanner reset()
```

返回值：扫描器的重置。

注意

调用此方法的形式为 scanner.reset()，该调用与以下调用的作用完全相同：

```
scanner.useDelimiter("\\p{javaWhitespace}+")
        .useLocale(Locale.getDefault())
        .useRadix(10);
```

示例

本示例创建一个扫描器，通过 reset()方法重置此扫描器。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.useRadix(16);
s.reset();
s.close();
```

3.18.33 skip 方法——忽略分隔符跳过匹配的输入信息

方法声明 1

在忽略分隔符的情况下跳过与指定模式匹配的输入信息。如果指定模式的固定匹配成功，此方法则跳过输入信息。

```
public Scanner skip(Pattern pattern)
```

入口参数：pattern 用于指定要跳过的模式。

返回值：此扫描器。

抛出异常

NoSuchElementException：如果未找到指定模式。

IllegalStateException：如果此扫描器已关闭。

注意

如果在当前位置未找到指定模式，则不跳过任何输入信息，并抛出 NoSuchElementException。由于此方法试图从扫描器的当前位置匹配指定模式，因此可与许多输入信息（例如".*"）匹配的模式可能会导致扫描器缓冲大量输入信息。注意，通过使用没有任何匹配项的模式，例如，sc.skip("[\t]*")，跳过某些信息同时不抛出 NoSuchElementException 是可能的。

示例

本示例创建一个扫描器，通过 skip()方法跳过输入信息。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.skip(Pattern.compile("\\d and"));
if(s.hasNext()){
    System.out.println(s.next());
}
s.close();
```

方法声明 2

跳过与从指定字符串构造的模式匹配的输入信息。

PDF电子书基地 http://dayo1982.400gb.com

此方法调用 skip(pattern)的行为与调用 skip(Pattern.compile(pattern))完全相同。

```
public Scanner skip(String pattern)
```

入口参数：pattern 指定要跳过的模式的字符串。

返回值：此扫描器。

抛出异常

IllegalStateException：如果此扫描器已关闭。

示例

本示例创建一个扫描器，通过 skip()方法跳过输入信息。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
s.skip("\\d and");
if(s.hasNext()){
    System.out.println(s.next());
}
s.close();
```

3.18.34 toString 方法——返回此 Scanner 的字符串表示形式

方法声明

返回此 Scanner 的字符串表示形式。Scanner 的字符串表示形式包含了对调试可能有用的信息。没有指定确切格式。

```
public String toString()
```

返回值：此扫描器的字符串表示形式。

示例

本示例创建一个扫描器，通过 toString()方法输出了此扫描器的字符串表示形式。

```
String input = "1 and 2 and yellow and blue";
Scanner s = new Scanner(input);
System.out.println(s.toString());
s.close();
```

3.18.35 useDelimiter 方法——设置为指定分隔模式

将此扫描器的分隔模式设置为指定模式。

方法声明 1

```
public Scanner useDelimiter(Pattern pattern)
```

入口参数：pattern 为分隔模式。

返回值：此扫描器。

示例

本示例创建了一个扫描器，调用 useDelimiter()方法设置为指定模式。

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter(Pattern.compile("\\sfish\\s"));
s.close();
```

方法声明 2

将此扫描器的分隔模式设置为从指定 String 构造的模式。

```
public Scanner useDelimiter(String pattern)
```

入口参数：pattern 为分隔模式的字符串。

返回值：此扫描器。

示例

本示例创建了一个扫描器，调用 useDelimiter()方法设置为指定模式。

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\sfish\\s");
s.close();
```

3.18.36 useLocale 方法——设置为指定的语言环境

方法声明

将此扫描器的语言环境设置为指定的语言环境。扫描器的语言环境会影响其与正则表达式匹配的默认基元的许多元素；参见上面的本地化数字。

```
public Scanner useLocale(Locale locale)
```

入口参数：locale 指定要使用的语言环境的字符串。

返回值：此扫描器。

示例

本示例创建一个扫描器，通过 useLocale()方法将此扫描器的语言环境设置为指定的英语语言环境。

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter(Pattern.compile("\\sfish\\s"));
s.useLocale(new Locale("en","US"));
System.out.println(s.locale());
s.close();
```

3.18.37 useRadix 方法——设置指定基数

方法声明

将此扫描器的默认基数设置为指定基数。扫描器的基数会影响其与正则表达式匹配的默认数的元素。

```
public Scanner useRadix(int radix)
```

入口参数：radix 为扫描数时要使用的基数。

返回值：此扫描器。

抛出异常：IllegalArgumentException，如果基数超出范围。

PDF电子书基地 http://dayo1982.400gb.com





示例

本示例创建一个扫描器，通过 useRadix()方法将此扫描器的默认基数设置为 16。

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter(Pattern.compile("\\sfish\\s"));
s.useRadix(16);
System.out.println(s.radix());
s.close();
```

3.19 Writer 类——写入字符流的抽象类

java.io 包中 Writer 类是写入字符流的抽象类。

3.19.1 append 方法——添加指定字符

该方法将指定的字符添加到 Writer 对象。



方法声明

```
public Writer append(char c) throws IOException
```

入口参数：c 是要添加的 16 位字符。

返回值：Writer 对象。

抛出异常：IOException。



示例

本示例实现创建 Writer 类的子类 PrintWriter 实例 out，并通过 append()方法向 Writer 类中添加字符。

```
Writer out=new PrintWriter(System.out);
out = out.append('a');
```

3.19.2 close 方法——关闭数据流

该方法实现关闭流。在通过 close()方法关闭流之后，调用 write()或 flush()方法将抛出 IOException 异常。关闭以前关闭的流无效。



方法声明

```
public abstract void close() throws IOException
```

抛出异常：IOException。



示例

本示例创建 Writer 类子类 PrintWriter 实例 out，并将其初始化为控制台的字符输出流，再定义一个字符数组，该数组是将要输出到控制台的字符串信息，并通过 out 对象在控制台输出该字符数组的内容。最后将流关闭。

```
Writer out=new PrintWriter(System.out);
char[] cs="本实例使用字符输出流，在控制台输出字符串".toCharArray();
out.write(cs);
out.close();
```

3.19.3 flush方法——刷新流的缓冲

该方法用于刷新流的缓冲。如果该流已保存缓冲区中各种write()方法的所有字符，调用flush()方法，会立即将它们写入预期目标。

方法声明

```
public abstract void flush() throws IOException
```

抛出异常：IOException。

示例

本示例创建Writer类子类PrintWriter实例out，并将其初始化为并将其初始化为控制台的字符输出流，再定义一个字符数组，该数组是将要输出到控制台的字符串信息，通过flush()方法刷新缓存。

```
Writer out=new PrintWriter(System.out);
char[] cs="本实例使用字符输出流，在控制台输出字符串".toCharArray();
out.write(cs);
out.flush();
out.close();
```

3.19.4 write方法——向字符输出流中写数据

该方法用于向字符输出流中写数据。

方法声明1

该方法将参数c转换为字节，然后写入输出流。

```
public void write(int c) throws IOException
```

入口参数：c指定要写入字符的int值。

抛出异常：IOException。

示例

本示例创建Writer类子类PrintWriter实例out，并将其初始化为并将其初始化为控制台的字符输出流，并通过write()方法向输出流中写入字符。由于数字97对应小写字母“a”的ASCII码值。因此，运行程序在控制台上输出小写字母“a”。

```
try {
Writer out=new PrintWriter(System.out);
out.write(97);
out.flush();
out.close();
} catch (IOException e) {
      e.printStackTrace();
}
```

方法声明2

将字符数组写入输出流。

PDF电子书基地 http://dayo1982.400gb.com

```
public void write(char[] chuf) throws IOException
```

入口参数：chuf 是要写入的字符数组。

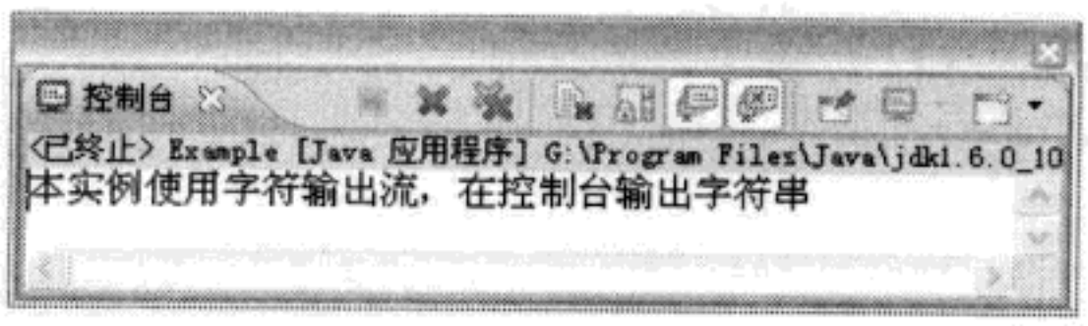
抛出异常：IOException。

示例

本示例创建 Writer 类子类 PrintWriter 实例 out，并将其初始化为控制台的字符输出流，通过 write()方法向流中写入字符数组。最后将流关闭。

```
try {
Writer out=new PrintWriter(System.out);
char[] cs="本实例使用字符输出流，在控制台输出字符串".toCharArray();
out.write(cs);
out.close();
} catch (IOException e) {
      e.printStackTrace();
}
```

本示例运行结果如下图所示。



字符输出流

方法声明 3

将参数数组中，指定索引的开始位置和指定长度写入字符输出流。

```
public void write(char[] chuf , int off , int len) throws IOException
```

入口参数

chuf：要写入流的字符数组。

off：要写入输出流的字符数组的开始索引位置。

len：要写入输出流的字符数组的长度。

抛出异常：IOException。

示例

本示例创建 Writer 类子类 PrintWriter 实例 out，并将其初始化为控制台的字符输出流。通过 write()方法向输出流中写入字符数组中索引位置从 0 开始的 5 个长度的元素。

```
try {
Writer out=new PrintWriter(System.out);
char[] cs="本实例使用字符输出流，在控制台输出字符串".toCharArray();
out.write(cs,0,5);
out.close();
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
```

方法声明 4

向字符输出流中写入字符串。

```
public void write(String str) throws IOException
```

入口参数：str 是要写入的字符串。

抛出异常：IOException。

示例

本示例创建 Writer 类子类 PrintWriter 实例 out，并将其初始化为并将其初始化为控制台的字符输出流，并通过 write()方法将字符串 str 写入字符输出流。

```
try {
Writer out=new PrintWriter(System.out);
String str = "Java 编程词典";
out.write(str);
out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

方法声明 5

向输出流中写入指定字符串参数的某一部分。

```
public void write(String str,int off,int len) throws IOException
```

入口参数

str：要写入的字符串。

off：要写入流的字符串的初始索引位置。

len：要写入流的字符串的长度。

抛出异常：IOException。

示例

本示例创建 Writer 类子类 PrintWriter 实例 out，并将其初始化为控制台的字符输出流。通过 write()方法将字符串 str 中，从索引位置 0 开始写入字符输出流，写入输出流的长度是 10。

```
try {
Writer out=new PrintWriter(System.out);
String str = "Java  编程词典 明日科技有限公司";
out.write(str,0,10);
out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```


PDF电子书基地 http://dayo1982.400gb.com

第4章 集 合 类

4.1 List 接口——有序集合类

List 接口位于 java.util 包中。List 集合包括 List 接口以及 List 接口的所有实现类。List 接口继承接口 Collection。List 接口属于列表类型，所以 List 接口还提供了一些适用于自身的常用方法。

4.1.1 add 方法——向列表中插入元素

add 方法用于向列表中插入元素。

方法声明 1

用于在列表的尾部插入指定元素。

```
boolean add(E e)
```

入口参数：要添加到列表中的元素。

返回值：如果 List 集合对象由于调用 add 方法而发生更改，则返回 true。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象，然后调用 add 方法，向该列表中添加数据。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                          //向列表中添加数据
    list.add("爱护地球");                          //向列表中添加数据
    list.add("从我做起");                          //向列表中添加数据
    //通过循环输出列表中的内容
    for(int i=0;i<list.size();i++){
        System.out.println(i+":"+list.get(i));
    }
}
```

示例运行结果如下。

```
0:保护环境
1:爱护地球
2:从我做起
```

方法声明 2

用于在列表的指定位置插入指定元素，并将当前处于该位置的元素及其后续元素的索引加 1。

```
void add(int index,
         E element)
```

入口参数：index，用于指定在其中插入指定元素处的索引；element 用于指定要插入的元素。

说明

索引值从 0 开始。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象，然后调用 add 方法，向该列表中添加数据。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                          //向列表中添加数据
    list.add("爱护地球");                          //向列表中添加数据
    list.add("从我做起");                          //向列表中添加数据
    list.add(1,"从我做起");                        //在第 1+1 个元素的位置添加数据
    //通过循环输出列表中的内容
    for(int i=0;i<list.size();i++){
        System.out.println(i+":"+list.get(i));
    }
}
```

示例运行结果如下。

```
0:保护环境
1:从我做起
2:爱护地球
3:从我做起
```

4.1.2 addAll 方法——将指定 collection 添加到列表中

addAll 方法用于将指定 collection 中的所有元素添加到列表中。

方法声明 1

用于将指定 collection 中的所有元素添加到列表的尾部。

```
boolean addAll(Collection<? extends E> c)
```

入口参数：用于指定要将全部元素添加到列表中的 collection。

返回值：如果 List 集合对象由于调用 addAll 方法而发生更改，则返回 true。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并调用 add 方法向该列表中添加数据，然后初始化一个列表对象 list_ad，并调用 add 方法向该列表中添加数据，再调用 addAll 方法将 list 添加到 list_ad 中，最后通过循环输出 list_ad。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
```


PDF电子书基地 http://dayo1982.400gb.com

```
        list.add("保护环境");                                    //向列表中添加数据
        list.add("爱护地球");                                    //向列表中添加数据
        list.add("从我做起");                                    //向列表中添加数据
        list.add(1,"从我做起");                                  //在第 1+1 个元素的位置添加数据
        List<String> list_ad=new ArrayList<String>();
        list_ad.add("公益广告");
        //将 list 中的全部元素添加到 list_ad 中
        System.out.println("添加是否成功："+list_ad.addAll(list));
        //通过循环输出列表中的内容
        for(int i=0;i<list_ad.size();i++){
            System.out.println(i+":"+list_ad.get(i));
        }
    }
```

示例的运行结果如下。

```
添加是否成功：true
0:公益广告
1:保护环境
2:从我做起
3:爱护地球
4:从我做起
```

方法声明 2

用于将 collection 中的所有元素添加到列表的指定位置。

```
boolean addAll(int index,
        Collection<? extends E> c)
```

入口参数：index 用于指定 collection 的第一个元素所插入位置的索引；c 用于指定要将全部元素添加到列表中的 collection。

返回值：如果 List 集合对象由于调用 addAll 方法而发生更改，则返回 true。

说明

索引值从 0 开始。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并调用 add 方法向该列表中添加数据，然后初始化一个列表对象 list_ad，并调用 add 方法向该列表中添加数据，再调用 addAll 方法将 list 添加到 list_ad 中，最后通过循环输出 list_ad。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                                    //向列表中添加数据
    list.add("爱护地球");                                    //向列表中添加数据
    list.add("从我做起");                                    //向列表中添加数据
    list.add(1,"从我做起");                                  //在第 1+1 个元素的位置添加数据
    List<String> list_ad=new ArrayList<String>();
```

```
        list_ad.add("公益广告");
        //将 list 中的全部元素添加到 list_ad 中
        System.out.println("添加是否成功："+list_ad.addAll(0,list));
        //通过循环输出列表中的内容
        for(int i=0;i<list_ad.size();i++){
            System.out.println(i+":"+list_ad.get(i));
        }
    }
```

示例运行结果如下。

```
添加是否成功：true
0:保护环境
1:从我做起
2:爱护地球
3:从我做起
4:公益广告
```

4.1.3 clear 方法——从列表中移除所有元素

clear 方法用于从列表中移除所有元素。

方法声明

```
void clear()
```

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并调用 add 方法向该列表中添加数据，然后通过循环输出 list 中的全部元素，最后调用 clear 方法移除所有元素，并输出 list 对象是否为空。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                          //向列表中添加数据
    list.add("爱护地球");                          //向列表中添加数据
    list.add("从我做起");                          //向列表中添加数据
    //通过循环输出列表中的内容
    for(int i=0;i<list.size();i++){
        System.out.println(i+":"+list.get(i));
    }
    list.clear();                                  //移除所有元素
    if(list.size()<=0){
        System.out.println("list 对象为空！");
    }
}
```

示例的运行结果如下。

```
0:保护环境
1:爱护地球
2:从我做起
list 对象为空！
```


PDF电子书基地 http://dayo1982.400gb.com

4.1.4 contains 方法——判断列表中是否包含指定元素

contains 方法用于判断列表中是否包含指定元素。

方法声明

```
boolean contains(Object o)
```

入口参数：要判断是否存在于列表中的元素。

返回值：true 或 false，如果列表中包含指定元素，则返回 true，否则返回 false。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并调用 contains 方法判断列表中是否包含“保护环境”的字符串。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                          //向列表中添加数据
    list.add("爱护地球");                          //向列表中添加数据
    list.add("从我做起");                          //向列表中添加数据
    //通过循环输出列表中的内容
    for(int i=0;i<list.size();i++){
        System.out.println(i+":"+list.get(i));
    }
    String o="保护环境";
    System.out.println("list 对象中是否包含元素 o： "+list.contains(o));
}
```

示例的运行结果如下。

```
0:保护环境
1:爱护地球
2:从我做起
list 对象中是否包含元素 o： true
```

4.1.5 containsAll 方法——判断列表中是否包含指定 collection 所有元素

containsAll 方法用于判断列表中是否包含指定 collection 的所有元素。

方法声明

```
boolean containsAll(Collection<?> c)
```

入口参数：要判断是否存在于列表中的 collection 对象。

返回值：true 或 false，如果列表中包含指定 collection 对象的所有元素，则返回 true，否则返回 false。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并

向该列表中添加 3 个元素，然后初始化一个列表对象 list_ad，并将 list 的全部元素添加到该列表中，再向该列表中添加一个元素“公益广告”，最后输出列表 list_ad 的全部内容，并调用 containsAll 方法判断列表中是否包含 list 的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                                    //向列表中添加数据
    list.add("爱护地球");                                    //向列表中添加数据
    list.add("从我做起");                                    //向列表中添加数据
    List<String> list_ad=new ArrayList<String>();
    list_ad.addAll(list);
    list_ad.add(0,"公益广告");
    //通过循环输出列表中的内容
    for(int i=0;i<list_ad.size();i++){
        System.out.println(i+":"+list_ad.get(i));
    }
    System.out.println("list_ad 中是否包含 list 的全部元素："+
                        list_ad.containsAll(list));
}
```

示例的运行结果如下。

```
0:公益广告
1:保护环境
2:爱护地球
3:从我做起
list_ad 中是否包含 list 的全部元素：true
```

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后初始化一个列表对象 list_ad，并向该列表中添加 3 个元素，最后调用 containsAll 方法判断列表 list_ad 中是否包含 list 的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                                    //向列表中添加数据
    list.add("爱护地球");                                    //向列表中添加数据
    list.add("从我做起");                                    //向列表中添加数据
    List<String> list_ad=new ArrayList<String>();
    list_ad.add("公益广告");
    list_ad.add("爱护地球");                                 //向列表中添加数据
    list_ad.add("从我做起");                                 //向列表中添加数据
    System.out.println("list_ad 中是否包含 list 的全部元素："+list_ad.containsAll(list));
}
```

示例的运行结果如下。

```
list_ad 中是否包含 list 的全部元素：false
```


PDF电子书基地 http://dayo1982.400gb.com

4.1.6 equals 方法——比较指定的对象与列表是否相等

equals 方法用于比较指定的对象与列表是否相等。

方法声明

```
boolean equals(Object o)
```

入口参数：与列表进行比较的对象。

返回值：true 或 false，如果指定对象与列表相同，则返回 true，否则返回 false。

示例

创建两个内容相同的 List 集合对象，然后使用 equals 方法进行判断。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                    //向列表中添加数据
    list.add("爱护地球");                    //向列表中添加数据
    list.add("从我做起");                    //向列表中添加数据
    List<String> list_ad=new ArrayList<String>();
    list_ad.add("保护环境");
    list_ad.add("爱护地球");                 //向列表中添加数据
    list_ad.add("从我做起");                 //向列表中添加数据
    boolean ret=list_ad.equals(list);
    if (ret) {
        System.out.println("两个 List 对象相同");
    } else {
        System.out.println("这不是两个相同的 List 对象");
    }
}
```

示例的运行结果如下。

```
两个 List 对象相同
```

4.1.7 get 方法——获取列表指定位置的元素

get 方法用于获取列表指定位置的元素。

方法声明

```
E get(int index)
```

入口参数：要返回元素的索引。

返回值：通过索引获取的元素。

说明

索引值从 0 开始。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该

列表中添加 3 个元素，然后调用 get 方法获取指定索引位置的元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    String ret=list.get(1);
    System.out.println("获取索引位置为 1 的元素："+ret);
}
```

示例的运行结果如下。

```
获取索引位置为 1 的元素：爱护地球
```

4.1.8 set 方法——替换列表中指定位置的元素

set 方法用于替换列表中指定位置的元素。

方法声明

```
E set(int index,
      E element)
```

入口参数：index 用于指定要替换元素所在位置的索引；element 用于指定要存储在指定位置的元素。

返回值：索引位置替换前的元素。

说明

索引值从 0 开始。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 set 方法将索引位置为 1 的元素替换为“少用塑料袋”，并输出索引位置为 1 的替换前的元素，最后输出列表中的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    String ret=list.set(1,"少用塑料袋");
    System.out.println("获取索引位置 1 替换前的内容："+ret);
    //通过循环输出列表中的内容
    for(int i=0;i<list.size();i++){
        System.out.println(i+":"+list.get(i));
    }
}
```

示例的运行结果如下。

```
获取索引位置 1 替换前的内容：爱护地球
0:保护环境
```


PDF电子书基地 http://dayo1982.400gb.com

1:少用塑料袋
2:从我做起

4.1.9 hashCode 方法——返回列表的哈希码值

hashCode 方法用于返回列表的哈希码值，对于一个列表，如果其中的元素不发生改变，则调用该方法返回的哈希码值是相同的，否则返回的哈希码值将发生改变。

说明

列表的哈希码定义为以下代码的计算结果。

```
int hashCode = 1;
    Iterator<E> i = list.iterator();
    while (i.hasNext()) {
        E obj = i.next();
        hashCode = 31*hashCode + (obj==null ? 0 : obj.hashCode());
    }
```

方法声明

```
int hashCode()
```

返回值：列表的哈希码值。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 hashCode 方法获取哈希码值并输出，再应用 set 方法修改索引值为 2 的元素，最后再调用 hashCode 方法获取哈希码值并输出。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");                          //向列表中添加数据
    list.add("爱护地球");                          //向列表中添加数据
    list.add("从我做起");                          //向列表中添加数据
    int ret=list.hashCode();
    System.out.println("list 对象的哈希码值为： "+ret);
    list.set(2, "从我做起！ ");
    System.out.println("将索引值为 2 的元素修改为“从我做起!” ");
    ret=list.hashCode();
    System.out.println("list 对象的哈希码值为： "+ret);
}
```

示例的运行结果如下。

```
list 对象的哈希码值为：-2004571724
将索引值为 2 的元素修改为“从我做起!”
list 对象的哈希码值为：-410272907
```

4.1.10 indexOf 方法——返回第一次出现指定元素的位置

indexOf 方法用于返回获取第一次出现指定元素的位置。

方法声明

```
int indexOf(Object o)
```

入口参数：要查找的元素。

返回值：如果在列表中存在指定的元素，将返回该元素第一次出现的位置，否则将返回-1。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 indexOf 方法获取“从我做起”在列表 list 中第一次出现的位置，并输出获取结果。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    int index=list.indexOf("从我做起");
    if(index==-1){
        System.out.println("该列表中不包括指定元素");
    }else{
        System.out.println("该元素第一次出现的位置为："+index);
    }
}
```

示例的运行结果如下。

```
该元素第一次出现的位置为：1
```

4.1.11 lastIndexOf 方法——返回最后一次出现指定元素的位置

lastIndexOf 方法用于返回最后一次出现指定元素的位置。

方法声明

```
int lastIndexOf(Object o)
```

入口参数：要查找的元素。

返回值：如果在列表中存在指定的元素，将返回该元素最后一次出现的位置，否则将返回-1。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 indexOf 方法获取“从我做起”在列表 list 中最后一次出现的位置，并输出获取结果。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
```


PDF电子书基地 http://dayo1982.400gb.com

```
        list.add("保护环境");              //向列表中添加数据
        list.add("从我做起");              //向列表中添加数据
        list.add("爱护地球");              //向列表中添加数据
        list.add("从我做起");              //向列表中添加数据
        int index=list.lastIndexOf("从我做起");
        if(index==-1){
            System.out.println("该列表中不包括指定元素");
        }else{
            System.out.println("该元素最后一次出现的位置为："+index);
        }
    }
```

示例的运行结果如下。

```
该元素最后一次出现的位置为：3
```

4.1.12 isEmpty 方法——判断集合是否为空

isEmpty 方法用于判断集合是否为空，如果列表不包含元素，则返回 true。

方法声明

```
boolean isEmpty()
```

返回值：如果在列表中不包含任何元素，将返回 true，否则返回 false。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 isEmpty 方法判断 list 对象是否为空，并输出判断结果，再执行 clear 方法从列表中移除所有元素，最后再次执行 isEmpty 方法判断 list 对象是否为空，并输出判断结果。

```
    public static void main(String[] args) {
        List<String> list=new ArrayList<String>();
        list.add("保护环境");              //向列表中添加数据
        list.add("从我做起");              //向列表中添加数据
        list.add("爱护地球");              //向列表中添加数据
        list.add("从我做起");              //向列表中添加数据
        boolean empty=list.isEmpty();
        if(empty){
            System.out.println("该列表为空");
        }else{
            System.out.println("该列表不为空");
        }
        list.clear();                      //从列表中移除所有元素
        System.out.println("执行 clear 方法后");
        empty=list.isEmpty();
        if(empty){
            System.out.println("该列表为空");
        }else{
            System.out.println("该列表不为空");
```

```
        }
    }
```

示例的运行结果如下。

```
该列表不为空
执行 clear 方法后
该列表为空
```

4.1.13 iterator 方法——返回迭代器

iterator 方法用于返回按适当顺序在列表的元素上进行迭代的迭代器。

方法声明

```
Iterator<E> iterator()
```

返回值：按适当顺序在列表的元素上进行迭代的迭代器。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 iterator 方法获取该列表的迭代器，最后通过循环输出该列表中的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    Iterator it=list.iterator();
    while(it.hasNext()){
        System.out.println(it.next());
    }
}
```

示例的运行结果如下。

```
保护环境
爱护地球
从我做起
```

4.1.14 listIterator 方法——返回列表迭代器

返回列表元素的列表迭代器。

方法声明 1

返回列表元素的列表迭代器（按适当顺序）。

```
ListIterator<E> listIterator()
```

返回值：此列表元素的列表迭代器（按适当顺序）。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象，向该列表

PDF电子书基地 http://dayo1982.400gb.com

中添加数据，调用 listIterator()方法返回此列表元素的列表迭代器。

```
List list = new ArrayList();
for (int i = 0; i < 5; i++) {
    list.add("编程词典" + i);
}
ListIterator li = list.listIterator();
while (li.hasNext()) {
    System.out.println(li.next());
}
```

示例运行结果如下。

```
编程词典 0
编程词典 1
编程词典 2
编程词典 3
编程词典 4
```

方法声明 2

返回列表中元素的列表迭代器（按适当顺序），从列表的指定位置开始。指定的索引表示 next 的初始调用所返回的第一个元素。previous 方法的初始调用将返回索引比指定索引少 1 的元素。

```
ListIterator<E> listIterator()
```

返回值：此列表中元素的列表迭代器（按适当顺序），从列表中的指定位置开始。

抛出异常

IndexOutOfBoundsException：如果索引超出范围（index < 0 || index > size()）。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象，向该列表中添加数据，调用 listIterator()方法从列表的指定位置开始，返回此列表元素的列表迭代器。

```
List list = new ArrayList();
for (int i = 0; i < 5; i++) {
    list.add("编程词典" + i);
}
ListIterator li = list.listIterator(2);
while (li.hasNext()) {
    System.out.println(li.next());
}
```

示例运行结果如下。

```
编程词典 2
编程词典 3
编程词典 4
```

4.1.15 remove 方法——移除列表中的指定元素

remove 方法用于移除列表中的指定元素。

方法声明 1

```
E remove(int index)
```

入口参数：要先移除元素的索引。

返回值：被移除的元素。

说明

索引值从 0 开始。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 remove 方法移除索引位置为 1 的元素，最后通过迭代输出该列表中的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    String str=list.remove(1);     //移除索引位置为 1 的元素
    System.out.println("被移除的元素为："+str);
    Iterator it=list.iterator();
    while(it.hasNext()){
        System.out.println(it.next());
    }
}
```

示例的运行结果如下。

```
被移除的元素为：爱护地球
保护环境
从我做起
```

方法声明 2

```
boolean remove(Object o)
```

入口参数：要移除的元素。

返回值：如果列表中包含要移除的元素，则返回 true，否则返回 false。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后调用 remove 方法移除元素“从我做起”，并输出执行结果。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
```


PDF电子书基地 http://dayo1982.400gb.com

```
list.add("保护环境");                          //向列表中添加数据
list.add("爱护地球");                          //向列表中添加数据
list.add("从我做起");                          //向列表中添加数据
boolean ret=list.remove("从我做起");           //移除指定元素
if(ret){
    System.out.println("元素被移除成功");
}else{
    System.out.println("列表中不包含此元素");
}
```

示例的运行结果如下。

```
保护环境
爱护地球
从我做起
```

4.1.16 removeAll 方法——从列表中移除指定 collection 包含的所有元素

removeAll 方法用于从列表中移除指定 collection 中包含的所有元素。

方法声明

```
boolean removeAll(Collection<?> c)
```

入口参数：包含从列表中移除元素的 collection 对象。

返回值：如果 List 集合对象由于调用 removeAll 方法而发生更改，则返回 true。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后再初始化一个列表对象 list1，并向该列表中添加两个与 list 列表中相同的元素，最后调用 removeAll 方法从列表 list1 中移除与列表 list 相同的元素，并输出执行结果。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");              //向列表中添加数据
    list.add("爱护地球");              //向列表中添加数据
    list.add("从我做起");              //向列表中添加数据
    List<String> list1=new ArrayList<String>();
    list1.add("保护环境");             //向列表中添加数据
    list1.add("爱护地球");             //向列表中添加数据
    boolean ret=list1.removeAll(list);
    if(ret){
        System.out.println("元素被移除成功");
    }else{
        System.out.println("列表中不包含要移除的元素");
    }
}
```

示例的运行结果如下。

```
元素被移除成功
```

4.1.17 retainAll 方法——保留指定 collection 包含的所有元素

retainAll 方法用于从列表中移除未包含在指定 collection 所有元素中的元素。

方法声明

```
boolean retainAll(Collection<?> c)
```

入口参数：包含在列表中保留元素的 collection 对象。

返回值：如果 List 集合对象由于调用 retainAll 方法而发生更改，则返回 true。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 3 个元素，然后再初始化一个列表对象 list1，并向该列表中添加两个与 list 列表中相同的元素，最后调用 retainAll 方法从列表 list 中移除与不包含在列表 list1 中的元素，并输出执行结果。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    List<String> list1=new ArrayList<String>();
    list1.add("保护环境");         //向列表中添加数据
    list1.add("爱护地球");         //向列表中添加数据
    boolean ret=list.retainAll(list1);
    if(ret){
        System.out.println("元素被移除成功");
    }else{
        System.out.println("列表中不包含要移除的元素");
    }
}
```

示例的运行结果如下。

```
元素被移除成功
```

4.1.18 size 方法——返回列表中元素的个数

size 方法用于返回列表中元素的个数。

方法声明

```
int size()
```

返回值：列表中的元素的个数。如果列表中元素的个数超过 2147483647，则返回 2147483647。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该

PDF电子书基地 http://dayo1982.400gb.com

列表中添加 3 个元素，然后通过 size 方法获取列表中元素的个数，并输出。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("保护环境");          //向列表中添加数据
    list.add("爱护地球");          //向列表中添加数据
    list.add("从我做起");          //向列表中添加数据
    int listSize=list.size();      //获取列表中元素的个数
    System.out.println("列表中元素的个数："+listSize);
}
```

示例的运行结果如下。

```
列表中元素的个数：3
```

4.1.19 subList 方法——获取列表中指定范围的子列表

subList 方法用于获取列表中指定范围的子列表，该列表支持原列表所支持的所有可选操作。

方法声明

```
List<E> subList(int fromIndex,
                int toIndex)
```

入口参数：fromIndex 用于指定新列表的起始点（包括该点），toIndex 用于指定新列表的结束点（不包括该点）。

返回值：列表中指定范围的子列表。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 6 个元素，然后调用 subList 方法获取该列表中索引值为 3～5 的元素组成的子列表，并输出。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("有些人生活太丰富了，");                    //向列表中添加数据
    list.add("太复杂了");                                //向列表中添加数据
    list.add("会忘记自己");                              //向列表中添加数据
    list.add("我是什么时候都知道，记住");                //向列表中添加数据
    list.add("自己是怎样简单的一个人");                  //向列表中添加数据
    list.add("我的颜色永远是灰色的…");                   //向列表中添加数据
    List<String> subList=list.subList(3, 5);             //获取子列表
    Iterator it=subList.iterator();
    while(it.hasNext()){
        System.out.println(it.next());
    }
}
```

示例的运行结果如下。

```
我是什么时候都知道，记住
自己是怎样简单的一个人
```

4.1.20 toArray 方法——返回所有元素的数组

toArray 方法用于按适当顺序返回包含列表中所有元素的数组。

方法声明 1

```
Object[] toArray()
```

返回值：由列表中所有元素组成的数组。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该列表中添加 5 个元素，然后调用 toArray 方法将该列表转换为数组，并通过循环输出该数组的全部元素。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("有些人生活太丰富了，");                    //向列表中添加数据
    list.add("太复杂了");                                //向列表中添加数据
    list.add("会忘记自己");                              //向列表中添加数据
    list.add("我是什么时候都知道，记住");                //向列表中添加数据
    list.add("自己是怎样简单的一个人");                  //向列表中添加数据
    Object[] arr=list.toArray();                         //转换为数组
    for(int i=0;i<arr.length;i++){
        System.out.println(i+"："+arr[i]);
    }
}
```

示例的运行结果如下。

```
0：有些人生活太丰富了，
1：太复杂了
2：会忘记自己
3：我是什么时候都知道，记住
4：自己是怎样简单的一个人
```

方法声明 2

```
<T> T[] toArray(T[] a)
```

入口参数：存储列表全部元素的数组，如果该数组不能容纳列表，将为此目的分配一个运行时类型相同的新数组。

返回值：包含数组中元素的数组。

说明

如果指定的数组能容纳列表，并有剩余空间（即数组的元素比列表的多），那么会将数组中紧随列表尾部的元素设置为 null。

示例

本示例使用 List 接口的实现类 ArrayList，初始化一个列表对象 list，并向该

PDF电子书基地 http://dayo1982.400gb.com

列表中添加 5 个元素，然后声明一个数组 a，再调用 toArray 方法将列表转换为数组，最后输出转换后的数组。

```
public static void main(String[] args) {
    List<String> list=new ArrayList<String>();
    list.add("有些人生活太丰富了，");                    //向列表中添加数据
    list.add("太复杂了");                                //向列表中添加数据
    list.add("会忘记自己");                              //向列表中添加数据
    list.add("我是什么时候都知道，记住");                //向列表中添加数据
    list.add("自己是怎样简单的一个人");                  //向列表中添加数据
    String[] a=new String[6];                            //声明一个数组
    Object[] arr=list.toArray(a);
    for(int i=0;i<arr.length;i++){
        System.out.println(i+"： "+arr[i]);
    }
}
```

示例的运行结果如下。

```
0：有些人生活太丰富了，
1：太复杂了
2：会忘记自己
3：我是什么时候都知道，记住
4：自己是怎样简单的一个人
5：null
```

4.2 Map 接口——映射集合类

Map 接口位于 java.util 包中。它是 Map 集合的接口定义，Map 集合使用键值对的方式实现数据保存，在 Map 集合中不能包含重复的键；每个键只能映射到一个值。Map 接口的常用实现类包括 HashMap、Hashtable、TreeMap 等集合类。

4.2.1 clear 方法——移除所有映射关系

该方法用于从 Map 集合中移除所有映射关系。该方法执行后，该集合对象的映射内容将为空。

方法声明

```
void clear()
```

示例

本示例使用 Map 接口的实现类 HashMap，初始化一个 Map 集合对象，然后调用 clear 方法，将该集合对象清空。

```
public static void main(String[] args) {
    Map map=new HashMap();
    map.put("昨天", "定制目录");
    map.put("今天", "开始写作");
```

```
        map.put("明天", "当然继续写作");
        System.out.println("map 大小为："+map.size());
        map.clear();        // 清除 map 集合的所有内容
        System.out.println("清除内容后的 map 大小是："+map.size());
    }
```

示例的运行结果如下。

```
map 大小为：3
清除内容后的 map 大小是：0
```

4.2.2 containsKey 方法——判断是否包含指定的键名

该方法用来判断 Map 集合对象中是否包含指定的键名。

方法声明

```
boolean containsKey(Object key)
```

入口参数：key 是要查询的 Map 集合的键名对象。

返回值：如果 Map 集合中包含指定的键名，那么返回 true，否则返回 false。

示例

本示例首先使用 HashMap 类创建 Map 集合对象，并向集合中添加几个元素，然后调用 containsKey 方法查询是否包含指定的键名。

```
public static void main(String[] args) {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    map.put("time", new Date());
    String key = "book";
    boolean contains = map.containsKey(key);
    if (contains) {
        System.out.println("在 Map 集合中包含键名" + key);
    } else {
        System.out.println("在 Map 集合中不包含键名" + key);
    }
}
```

示例的运行结果如下。

```
在 Map 集合中包含键名 book
```

4.2.3 containsValue 方法——判断是否包含指定的键值

该方法用来判断 Map 集合中是否包含指定的键值。

方法声明

```
boolean containsValue(Object value)
```

入口参数：value 是要查询的 Map 集合的指定键值对象。

返回值：如果 Map 集合中包含要查询的键值对象，返回 true，否则返回 false。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例首先使用 HashMap 类创建 Map 集合对象，并向集合中添加几个元素，然后调用 containsValue 方法查询是否包含指定的键值对象。

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    map.put("time", new Date());
    Thread.sleep(1000);                    // 休眠 1 秒，使两个 Date 对象不相等
    Date value = new Date();
    boolean contains = map.containsValue(value);
    if (contains) {
        System.out.println("在 Map 集合中包含键名" + value);
    } else {
        System.out.println("在 Map 集合中不包含键名" + value);
    }
}
```

示例的运行结果如下。

```
在 Map 集合中不包含键名 Tue Dec 02 10:17:48 CST 2008
```

4.2.4 equals 方法——判断是否与指定的对象相同

该方法用来判断 Map 集合是否与指定的对象相同，这个对象通常情况下也是一个集合。

方法声明

```
boolean equals(Object o)
```

入口参数：o 是要比较的对象，通常情况下这个对象也是一个 Map 集合。

返回值：如果 Map 集合与比较的对象相同，则返回 true；否则返回 false。

示例

创建两个内容相同的 Map 集合对象，然后使用 equals 方法进行判断。

```
public static void main(String[] args) throws InterruptedException {
    Map map1 = new HashMap();
    map1.put("apple", "新鲜的苹果");
    map1.put("computer", "配置优良的计算机");
    map1.put("book", "堆积成山的图书");
    Map map2 = new HashMap();
    map2.put("apple", "新鲜的苹果");
    map2.put("computer", "配置优良的计算机");
    map2.put("book", "堆积成山的图书");
    boolean contains = map1.equals(map2);
    if (contains) {
        System.out.println("两个 Map 对象相同");
```

```
        } else {
            System.out.println("这不是两个相同的 Map 对象");
        }
    }
```

示例的运行结果如下。

```
两个 Map 对象相同
```

4.2.5 get 方法——返回指定键所映射的值

该方法返回指定键所映射的值。

方法声明

```
V get(Object key)
```

入口参数：key 是指定的 Map 集合中的键名。

返回值：使用指定的 key 从 Map 集合中获取的键值对象。

示例

本示例使用 HashMap 类创建 Map 集合对象，并初始化一定数量的集合内容。然后使用 get 方法获取指定键名的键值对象。

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    Object get = map.get("apple");
    if (get instanceof String) {
        String value = (String) get;
        System.out.println("在 Map 集合中键名 apple 的键值是：" + value);
    }
}
```

示例运行结果如下。

```
在 Map 集合中键名 apple 的键值是：新鲜的苹果
```

4.2.6 isEmpty 方法——判断是否为空

该方法用来判断 Map 集合对象是否包含内容，也就是判断该 Map 集合容器是不是空的。

方法声明

```
boolean isEmpty()
```

返回值：如果 Map 集合对象不包含任何内容，返回 true，否则返回 false。

示例

初始化一个Map集合对象，分别在添加内容到Map集合的前后，输出 isEmpty 方法的返回值。

PDF电子书基地 http://dayo1982.400gb.com

```
public static void main(String[] args) throws InterruptedException {
    System.out.println("创建 Map 集合对象");
    Map map = new HashMap();
    System.out.println("Map 集合的 isEmpty 方法返回值是："+map.isEmpty());
    System.out.println("添加内容到 Map 集合");
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    System.out.println("Map 集合的 isEmpty 方法返回值是："+map.isEmpty());
}
```

示例运行结果如下。

```
创建 Map 集合对象
Map 集合的 isEmpty 方法返回值是：true
添加内容到 Map 集合
Map 集合的 isEmpty 方法返回值是：false
```

4.2.7 keySet 方法——获取 Map 集合的所有 key

该方法将获取 Map 集合的所有 key（键名），并存放在一个 Set 集合对象中。

方法声明

```
Set<K> keySet()
```

返回值：包含 Map 集合对象的所有键名的 Set 集合对象。

示例

获取 Map 对象的所有键名，然后迭代输出所有键名。

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    map.put("time", new Date());
    Set keySet = map.keySet();
    for (Object keyName : keySet) {
        System.out.println("键名： " + keyName);
    }
}
```

示例运行结果如下。

```
键名： computer
键名： time
键名： apple
键名： book
```

4.2.8 put 方法——向指定索引位置添加对象

该方法用来向指定索引位置添加对象。其他对象的索引相对后移一位。

方法声明

```
V put(K key, V value)
```

入口参数

key：是要保存到 Map 集合中的键名。

value：是要保存到 Map 集合中对应键名的键值对象。

返回值：以前与 key 关联的值。

注意

对于返回值，如果没有针对 key 的映射关系，则返回 NULL（如果实现 Map 集合的类支持 NULL 值，那么返回的 NULL 值也可能表示此映射以前将 NULL 值与 key 关联）。

示例

本示例创建并初始化一个 Map 集合对象，并调用 put 方法行集合对象中插入多个内容。最后输出 Map 集合的大小，也就是集合包含内容的数量。

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    map.put("time", new Date());
    int size=map.size();
    System.out.println("Map 集合的大小为："+size);
}
```

示例运行结果如下。

```
Map 集合的大小为：4
```

4.2.9 putAll 方法——追加另一个 Map 对象到当前 Map 集合

该方法用来追加另一个 Map 对象到当前 Map 集合对象中，它会把另一个 Map 集合对象中的所有内容，添加到当前 Map 集合对象中。

方法声明

```
void putAll(Map<? extends K,? extends V> m)
```

入口参数：m 是一个 Map 集合对象。

示例

创建一个 Map 集合对象，为它添加一些内容并输出该集合的大小，然后创建第二个 Map 集合对象，也添加一些内容，输出集合大小，最后把第二个 Map 集合添加到第一个 Map 集合对象中，再次输出第一个集合的大小。

```
public static void main(String[] args) {
    Map map1 = new HashMap();
```


PDF电子书基地 http://dayo1982.400gb.com

```
        map1.put("apple", "新鲜的苹果");
        map1.put("computer", "配置优良的计算机");
        map1.put("book", "堆积成山的图书");
        System.out.println("第一个 Map 集合大小为： "+map1.size());
        Map map2 = new HashMap();
        map2.put("apple2", "新鲜的苹果");
        map2.put("computer2", "配置优良的计算机");
        map2.put("book", "堆积成山的图书");
        System.out.println("第二个 Map 集合大小为： "+map2.size());
        System.out.println("把第二个 Map 集合添加到第一个 Map 集合中");
        map1.putAll(map2);
        System.out.println("整合后的第一个 Map 集合大小为： "+map1.size());
    }
```

示例运行结果如下。

```
第一个 Map 集合大小为：3
第二个 Map 集合大小为：3
把第二个 Map 集合添加到第一个 Map 集合中
整合后的第一个 Map 集合大小为：5。
```

注意

整合后的 Map 集合大小是 5 而不是 6，那是因为两个 Map 集合中有一个重复的键名“book”，Map 集合的键名是不能重复的，所以新的“book”键值，取代了旧的“book”键值。

4.2.10 remove 方法——移除 Map 集合中指定键名的内容

该方法用于移除 Map 集合中指定键名的内容。

方法声明

```
V remove(Object key)
```

入口参数：key 是要移除的 Map 集合中的键名。

返回值：以前与 key 指定的键名关联的键值。

示例

创建一个 Map 集合，向该集合添加一些内容，然后使用 remove 方法移除部分内容。

```
    public static void main(String[] args) {
        Map map = new HashMap();
        map.put("apple", "新鲜的苹果");
        map.put("computer", "配置优良的计算机");
        map.put("book", "堆积成山的图书");
        System.out.println("Map 集合大小为： "+map.size());
        map.remove("apple");
        map.remove("book");
        System.out.println("移除内容后的 Map 集合大小为： "+map.size());
    }
```

示例运行结果如下。

```
Map 集合大小为：3
移除内容后的 Map 集合大小为：1
```

4.2.11 size 方法——获取 Map 集合类的大小

该方法用于获取 Map 集合类的大小，也就是 Map 集合包含键值对关系映射的数量。

方法声明

```
int size()
```

返回值：Map 集合中包含的关系映射的数量。

示例

创建一个 Map 集合对象，并添加一些内容，然后调用 size 方法获取 Map 集合对象的大小。

```
public static void main(String[] args) {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    System.out.println("Map 集合大小为："+map.size());
}
```

示例运行结果如下。

```
Map 集合大小为：3
```

4.2.12 values 方法——获取 Map 集合中所有键值对象

该方法用来获取 Map 集合中的所有键值对象，这些键值对象将存放在另一个集合对象中。

方法声明

```
Collection<V> values()
```

返回值：包含 Map 集合中所有键值的集合对象。

示例

创建一个 Map 集合对象，添加一些内容，然后调用 values 方法获取包含所有键值的集合对象，并输出这些键值信息。

```
public static void main(String[] args) {
    Map map = new HashMap();
    map.put("apple", "新鲜的苹果");
    map.put("computer", "配置优良的计算机");
    map.put("book", "堆积成山的图书");
    Collection values = map.values();
    for (Object object : values) {
```


PDF电子书基地 http://dayo1982.400gb.com

```
            System.out.println("键值： " + object.toString());
        }
    }
```

示例的运行结果如下。

```
键值：配置优良的计算机
键值：新鲜的苹果
键值：堆积成山的图书
```

4.3 Set 接口——无重复元素集合类

Set 集合位于 Java.util 包中，它的特点是集合中的对象不按特定方式排序，并且没有重复对象。它的有些实现类能对集合中的对象按特定方法排序。常用的 Set 接口实现类有 HashSet 和 LinkedHashSet。

HashSet 类不保证容器的迭代顺序，特别是它不保证该顺序恒久不变，此类允许使用 NULL 元素。

LinkedHashSet 类继承于 HashSet 类，此类与 HashSet 类的不同之外在于，它维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，即按照将元素插入到集合中的顺序（插入顺序）进行迭代。但是，插入顺序不受在集合中重新插入的元素的影响。

4.3.1 add 方法——向 Set 集合中添加对象

该方法用来向 Set 集合添加对象。如果 Set 集合中已经包含相同的对象，则不改变 Set 集合。

方法声明

```
boolean add(E e)
```

入口参数：e 是要添加到 Set 集合中的对象。

返回值：如果 Set 集合中不包含要添加的对象，则添加对象并返回 true，否则返回 false。

示例

使用 HashSet 类创建 Set 集合对象，然后调用 add 方法为 Set 集合添加内容。

```
public static void main(String[] args) {
        Set set=new HashSet();
        set.add(new Date());
        set.add("apple");
        set.add("book");
        set.add(new Integer(3));
        set.add(new Socket());
        int size=set.size();
        System.out.println("Set 集合的大小为： " + size);
}
```

4.3.2 addAll 方法——向 Set 集合添加另一个集合的所有内容

该方法用来向 Set 集合添加另一个集合对象所包含的所有内容。

方法声明

```
boolean addAll(Collection<? extends E> c)
```

入口参数：c 是要添加到 Set 集合中的另一个集合对象。

返回值：如果 Set 集合发生更改，则返回 true。

示例

创建一个 Set 集合，为该集合添加一些对象，然后在创建第二个 Set 集合对象，同样添加一些对象，然后调用 addAll 方法将第二个 Set 集合添加到第一个 Set 集合中，并且分别输出调用 addAll 方法前后 Set 集合的大小。

```
public static void main(String[] args) {
    Set set1=new HashSet();
    set1.add(new Date());
    set1.add("apple");
    set1.add(new Integer(3));
    set1.add(new Socket());
    int size=set1.size();
    System.out.println("Set1 集合的大小为：" + size);
    Set set2=new HashSet();
    set2.add("book");
    set2.add(new Long(3));
    int size2=set2.size();
    System.out.println("Set2 集合的大小为：" + size2);
    set1.addAll(set2);
    System.out.println("合并后 Set1 集合的大小为：" + set1.size());
}
```

示例的运行结果如下。

```
Set1 集合的大小为：4
Set2 集合的大小为：2
合并后 Set1 集合的大小为：6
```

示例

将 List 集合添加到 Set 集合中。

```
public static void main(String[] args) {
    Set set=new HashSet();
    set.add(new Date());
    set.add("apple");
    set.add(new Socket());
    int size=set.size();
    System.out.println("Set 集合的大小为：" + size);
    List list=new ArrayList();
```


PDF电子书基地 http://dayo1982.400gb.com

```
        list.add("图书");
        list.add(new Long(3));
        int size2=list.size();
        System.out.println("List 集合的大小为：" + size2);
        set.addAll(list);
        System.out.println("合并后 Set 集合的大小为：" + set.size());
    }
```

示例运行结果如下。

```
Set 集合的大小为：3
List 集合的大小为：2
合并后 Set 集合的大小为：5
```

4.3.3 clear 方法——从 Set 集合中移除所有内容

该方法用于从 Set 集合中移除所有内容。该方法执行后，Set 集合对象不再包含任何对象。

方法声明

```
void clear()
```

示例

本示例使用 HashSet 类构建 Set 集合对象，分别输出调用 clear 方法前后 Set 集合的大小。

```
public static void main(String[] args) {
        Set set=new HashSet();
        set.add(new Date());
        set.add("apple");
        set.add(new Socket());
        int size=set.size();
        System.out.println("Set 集合的大小为：" + size);
        set.clear();
        System.out.println("执行 clear 方法后 Set 集合的大小为：" + set.size());
    }
```

示例运行结果如下。

```
Set 集合的大小为：3
执行 clear 方法后 Set 集合的大小为：0
```

4.3.4 contains 方法——判断 Set 集合是否包含指定对象

该方法用来判断 Set 集合是否包含指定的对象。

方法声明

```
boolean contains(Object o)
```

入口参数：o 是要查询的对象。

返回值：如果 Set 集合包含指定的对象，则返回 true，否则返回 false。

示例

本示例使用HashSet类构建Set集合对象，并添加一些内容，然后使用contains方法判断 Set 集合是否包含指定的对象。

```
public static void main(String[] args) {
    Set set = new HashSet();
    set.add(new Date());
    set.add("apple");
    set.add(new Socket());
    boolean contains = set.contains("apple");
    if (contains) {
        System.out.println("Set 集合包含字符串 apple");
    } else {
        System.out.println("Set 集合不包含字符串 apple");
    }
}
```

示例运行结果如下。

```
Set 集合包含字符串 apple
```

4.3.5 containsAll 方法——判断 Set 集合是否包含另一个集合中全部对象

该方法用来判断 Set 集合是否包含另一个集合中的全部内容。

方法声明

```
boolean containsAll(Collection<?> c)
```

入口参数：c 是要判断的另一个集合对象。

返回值：如果 Set 集合包含参数 c 指定的集合对象中所有内容，则返回 true，否则返回 false。

示例

本示例首先创建 Set 集合实例，并添加一些对象。然后创建包含和 Set 集合部分内容相同的 List 集合，调用 containsAll 方法判断 Set 集合是否包含 List 集合的内容。

```
public static void main(String[] args) {
    Set set = new HashSet();
    set.add(new Date());
    set.add("apple");
    set.add(new Socket());
    set.add("TV");
    List list=new ArrayList();
    list.add("apple");
    list.add("TV");
    boolean contains = set.containsAll(list);
    if (contains) {
        System.out.println("Set 集合包含 List 集合的内容");
```


PDF电子书基地 http://dayo1982.400gb.com

```
        } else {
            System.out.println("Set 集合不包含 List 集合的内容");
        }
    }
```

示例运行结果如下。

```
Set 集合包含 List 集合的内容
```

4.3.6 equals 方法——比较指定对象与此 Set 是否相等

该方法用来判断当前 Set 集合与另一个 Set 集合对象是否相等。

方法声明

```
boolean equals(Object o)
```

入口参数：o 是要和 Set 集合对象进行判断的对象。

返回值：如果参数 o 是 Set 集合对象并且和当前 Set 集合内容相同，则返回 true，否则返回 false。

示例

本实例演示 Set 集合与另一个内容数量不相等的 Set 集合的 Equals 判断。

```
public static void main(String[] args) {
    Set set1 = new HashSet();
    set1.add(new Date());
    set1.add("apple");
    set1.add(new Socket());
    set1.add("TV");
    Set se2=new HashSet();
    se2.add("apple");
    se2.add("TV");
    boolean contains = set1.equals(se2);
    if (contains) {
        System.out.println("两个 Set 集合相同");
    } else {
        System.out.println("Set1 集合与 Set2 集合不相同");
    }
}
```

示例运行结果如下。

```
Set1 集合与 Set2 集合不相同
```

4.3.7 isEmpty 方法——判断 Set 集合是否为空

该方法用来判断 Set 集合对象是否包含内容，也就是判断该 Set 集合容器是不是空的。

方法声明

```
boolean isEmpty()
```

返回值：如果 Set 集合对象不包含任何内容，返回 true，否则返回 false。

示例

初始化一个 Set 集合对象，分别在添加内容到 Set 集合的前后，输出 isEmpty 方法的返回值。

```
public static void main(String[] args) throws InterruptedException {
    System.out.println("创建 Set 集合对象");
    Set set = new HashSet();
    System.out.println("Set 集合的 isEmpty 方法返回值是："+set.isEmpty());
    System.out.println("添加内容到 Set 集合");
    set.add("apple");
    set.add("配置优良的计算机");
    set.add("book");
    System.out.println("Set 集合的 isEmpty 方法返回值是："+set.isEmpty());
}
```

示例运行结果如下。

```
创建 Set 集合对象
Set 集合的 isEmpty 方法返回值是：true
添加内容到 Set 集合
Set 集合的 isEmpty 方法返回值是：false
```

4.3.8 iterator 方法——获取 Set 集合的迭代器

该方法用来获取 Set 集合的迭代器，这个迭代器是 Iterator 接口的实现，它能够遍历 Set 集合。

方法声明

```
Iterator<E> iterator()
```

返回值：Iterator 迭代对象。

示例

本示例首先创建 Set 集合对象，添加一些内容到 Set 集合，然后调用 iterator 方法获取 Set 集合的迭代器，使用这个迭代器遍历 Set 集合所有内容。

```
public static void main(String[] args) throws InterruptedException {
    Set set = new HashSet();
    set.add("apple");
    set.add("配置优良的计算机");
    set.add("book");
    Iterator iterator = set.iterator();
    while (iterator.hasNext()) {            // 遍历 Set 集合
        Object object = iterator.next();
        System.out.println(object.toString());
    }
}
```

示例运行结果如下。

```
apple
book
配置优良的计算机
```


PDF电子书基地 http://dayo1982.400gb.com

4.3.9 remove 方法——移除 Set 集合中指定对象

该方法用于移除 Set 集合中的指定对象。

方法声明

```
boolean remove(Object o)
```

入口参数：o 是要移除的对象。

返回值：如果 Set 集合包含参数 o 指定的对象，则返回 true，否则返回 false。

示例

本示例创建了 Set 集合对象，然后调用 remove 方法移除指定对象。

```
public static void main(String[] args) throws InterruptedException {
    Set set = new HashSet();
    set.add("apple");
    set.add("配置优良的计算机");
    set.add("book");
    System.out.println("Set 大小为：  "+set.size());
    set.remove("book");
    System.out.println("从 Set 集合移除了\"book\"");
    System.out.println("Set 大小为：  "+set.size());
}
```

示例运行结果如下。

```
Set 大小为：3
从 Set 集合移除了"book"
Set 大小为：2
```

4.3.10 removeAll 方法——移除另一个集合所包含的所有内容

该方法用于从 Set 集合移除另一个集合对象所包含的所有内容。

方法声明

```
boolean removeAll(Collection<?> c)
```

入口参数：c 是另一个集合对象，可以是 Set 集合或者 List 集合等。

返回值：如果 Set 集合因调用本方法而改变，则返回 true。

示例

本示例创建了 Set 集合对象，然后调用 removeAll 方法移除另一个集合包含的对象。

```
public static void main(String[] args) {
    Set set1 = new HashSet();
    set1.add(new Date());
    set1.add("apple");
    set1.add(new Socket());
    set1.add("TV");
    System.out.println("Set1 的大小为：  "+set1.size());
```

```
        Set set2=new HashSet();
        set2.add("apple");
        set2.add("TV");
        System.out.println("Set2 的大小为："+set2.size());
        set1.removeAll(set2);
        System.out.println("从 Set1 集合移除 Set2 集合包含的内容");
        System.out.println("Set1 的大小为："+set1.size());
    }
```

示例运行结果如下。

```
Set1 的大小为：4
Set2 的大小为：2
从 Set1 集合移除 Set2 集合包含的内容
Set1 的大小为：2
```

4.3.11 retainAll 方法——保留另一个集合所包含的所有内容

该方法将保留与指定集合中的相同对象，而删除不相同的对象，它和 removeAll 方法正好相反。

方法声明

```
boolean retainAll(Collection<?> c)
```

入口参数：c 是另一个集合对象，可以是 Set 集合或者 List 集合等。

返回值：如果 Set 集合因调用本方法而改变，则返回 true。

示例

本示例创建了 Set 集合对象，然后调用 retainAll 方法移除与另一个集合中不相同的内容。

```
public static void main(String[] args) {
        Set set1 = new HashSet();
        set1.add(new Date());
        set1.add("apple");
        set1.add(new Socket());
        set1.add("TV");
        System.out.println("Set1 的大小为："+set1.size());
        Set set2=new HashSet();
        set2.add("TV");
        System.out.println("Set2 的大小为："+set2.size());
        set1.retainAll(set2);
        System.out.println("从 Set1 集合移除 Set2 集合包含的内容");
        System.out.println("Set1 的大小为："+set1.size());
    }
```

示例运行结果如下。

```
Set1 的大小为：4
Set2 的大小为：1
从 Set1 集合移除 Set2 集合包含的内容
Set1 的大小为：1
```


PDF电子书基地 http://dayo1982.400gb.com

4.3.12 size 方法——获取 Set 集合类的大小

该方法用于获取 Set 集合类的大小，也就是 Set 集合包含的对象的数量。

方法声明

```
int size()
```

返回值：Set 集合中包含的对象的数量。

示例

创建一个 Set 集合对象，并添加一些内容，然后调用 size 方法获取 Set 集合对象的大小。

```
public static void main(String[] args) {
    Set set = new HashSet();
    set.add("apple");
    set.add("computer");
    set.add("book");
    set.add(new Date());
    System.out.println("Set 集合大小为："+set.size());
}
```

示例运行结果如下。

```
Set 集合大小为：4
```

4.3.13 toArray 方法——用 Set 集合的所有对象创建数组

该方法可以用 Set 集合中的所有对象创建一个数组。

方法声明 1

根据 Set 集合的大小，生成相同长度的数组，该数组包含了 Set 集合中的所有内容。

```
Object[] toArray()
```

返回值：包含 Set 集合所有内容的数组。

示例

把 Set 集合中的所有内容保存到一个新的数组中。

```
public static void main(String[] args) {
    Set set = new HashSet();
    set.add("apple");
    set.add("computer");
    set.add("book");
    set.add(new Date());
    Object[] toArray = set.toArray();
    System.out.println("数组的长度是："+toArray.length);
}
```

示例运行结果如下。

```
数组的长度是：4
```

方法声明 2

使用指定的数组，存储 Set 集合中的所有内容。

```
<T> T[] toArray(T[] a)
```

入口参数：a 是保存 Set 集合所有内容的数组。

返回值：包含 Set 集合所有内容的数组。

注意

对于返回值，如果参数指定的数组能够容纳 Set 集合的所有内容，就使用该数组保存 Set 集合中的所有对象，并返回该数组；否则，返回一个新的能够容纳 Set 集合中所有内容的数组。

如果参数指定的数组长度大于 Set 集合的大小，那么数组的剩余空间全部复制为 NULL 值。

示例

把 Set 集合的所有内容输出到超出 Set 集合大小的数组中。

```
public static void main(String[] args) {
    Set set = new HashSet();
    set.add("apple");
    set.add("computer");
    set.add("book");
    set.add("String 也是对象，不是基本数据类型");
    String[] strArray = new String[6];
    String[] toArray = (String[]) set.toArray(strArray);
    System.out.println("数组的长度是：" + toArray.length);
    for (String string : toArray) {
        System.out.println(string);
    }
}
```

示例运行结果如下。

```
数组的长度是：6
computer
String 也是对象，不是基本数据类型
apple
book
null
null
```


PDF电子书基地 http://dayo1982.400gb.com

第 5 章　数据库编程

5.1　DriverManager 类——驱动程序管理类

DriverManager 类位于 java.sql 包中，是用来管理 JDBC 驱动程序的基本服务。作为初始化的一部分，它会加载系统属性中引用的驱动程序类。它可以跟踪可用的驱动程序，注册、注销以及为数据库连接选择合适的驱动程序，设置登录时间限制等，并在数据库的驱动程序之间建立连接。

5.1.1　deregisterDriver 方法——删除一个驱动程序

从 DriverManager 的列表中删除一个驱动程序。applet 只能注销取自其自身的类加载器的驱动程序。

方法声明

```
public static void deregisterDriver(Driver driver) throws SQLException
```

入口参数：driver 要删除的 JDBCDriver。

抛出异常：SQLException。

示例

本示例注册了一个 mysql 的驱动，然后调用 deregisterDriver()方法将其删除。

```
Class.forName("com.mysql.jdbc.Driver");                    //注册 mysql 驱动
String url = "jdbc:mysql://localhost:3306/test?user=root&password=111";
Driver driver = DriverManager.getDriver(url);
DriverManager.deregisterDriver(driver);                    删除 mysql 驱动
```

5.1.2　getDriver 方法——选择一个适当的驱动程序

试图查找能理解给定 URL 的驱动程序。DriverManager 试图从已注册的 JDBC 驱动程序集中选择一个适当的驱动程序。

方法声明

```
public static Driver getDriver(String url) throws SQLException
```

入口参数：url dbc:subprotocol:subname 形式的数据库 URL。

返回值：表示可以连接到给定 URL 的驱动程序的 Driver 对象。

抛出异常：SQLException。

示例

本示例注册了一个 mysql 的驱动，通过 DriverManager 的 getDriver()方法返回了 URL 的驱动程序。

```
Class.forName("com.mysql.jdbc.Driver");                    //注册 mysql 驱动
String url = "jdbc:mysql://localhost:3306/test?user=root&password=111";
Driver driver = DriverManager.getDriver(url);
```

5.1.3　getDrivers 方法——获取全部已加载的驱动程序

获取带有当前调用者可以访问的所有当前已加载 JDBC 驱动程序的 Enumeration。

方法声明

```
public static Enumeration<Driver> getDrivers()
```

返回值：由调用者的类加载器加载的 JDBCDriver 组成的列表。

示例

本示例注册了一个 mysql 的驱动，通过 DriverManager 的 getDrivers()方法返回了一个 Enumeration，并输出了所有的驱动程序的类名称。

```
try {
    Class.forName("com.mysql.jdbc.Driver");          //注册 mysql 驱动
    Enumeration em = DriverManager.getDrivers();
    while(em.hasMoreElements()) {
        Driver d =(Driver) em.nextElement();
        System.out.println(d.getClass().getName());
    }
} catch(Exception e) {
    e.printStackTrace();
}
```

5.1.4　getConnection 方法——获取数据库连接

该方法用于获得试图建立到指定数据库 url 的连接。DriverManager 试图从已注册的 JDBC 驱动程序集中选择一个适当的驱动程序。

方法声明 1

```
public static Connection getConnection(String url)
                                    throws SQLException
```

入口参数：url 是访问数据库的 URL 路径。

返回值：连接到数据库的 Connection 对象。

抛出异常：SQLException。

示例

下面的代码利用 getConnection 方法创建与 MySQL 数据库的连接，并返回连

PDF电子书基地 http://dayo1982.400gb.com

接对象。

```
public Connection getConnection(){
    Connection con=null;
     try{
       Class.forName("com.mysql.jdbc.Driver");                    //注册数据库驱动
       String url="jdbc:mysql://localhost:3306/test?user=root&password=root";
       con=DriverManager.getConnection(url);
       System.out.println("数据库连接成功！ ");
    }catch(Exception e){
        e.printStackTrace();
    }
    return con;                                                   //返回一个连接
}
```

方法声明 2

```
public static Connection getConnection(String url,
                                       Properties info)
                                throws SQLException
```

入口参数

url：访问数据库的 URL 路径。

Info：一个持久的属性集对像。应包括 user 和 password 属性。

返回值：连接到数据库的 Connection 对象。

抛出异常：SQLException。

示例

下面的代码利用 getConnection 方法创建与 MySQL 数据库的连接，并返回连接对象。

```
public Connection getConnection(){
    Connection con=null;
    Properties info = new Properties();
        info.setProperty("user","root");
        info.setProperty("password","root");
     try{
        Class.forName("com.mysql.jdbc.Driver");                   //注册数据库驱动
        String url="jdbc:mysql://localhost:3306/test";            //test 为数据库名称
        con=DriverManager.getConnection(url,info);
        System.out.println("数据库连接成功！ ");
    }catch(Exception e){
        e.printStackTrace();
    }
    return con;                                                   //返回一个连接
}
```

方法声明 3

```
public static Connection getConnection(String url,
                                       String user,
                                       String password)
                                throws SQLException
```

入口参数

url：访问数据库的 URL 路径。

user：连接数据库的用户名。

password：连接数据库的密码。

返回值：连接到数据库的 Connection 对象。

抛出异常：SQLException。

示例

下面的代码利用 getConnection 方法创建与 MySQL 数据库的连接，并返回连接对象。

```
public Connection getConnection(){
    Connection con=null;
     try{
         Class.forName("com.mysql.jdbc.Driver");              //注册数据库驱动
         String url="jdbc:mysql://localhost:3306/test";       //test 为数据库名
         String user = "root";                                //数据库连接用户名
         String password = "root";                            //数据库连接密码
         con=DriverManager.getConnection(url,user,password);
         System.out.println("数据库连接成功！ ");
    }catch(Exception e){
         e.printStackTrace();
    }
    return con;                                               //返回一个连接
}
```

5.1.5 getLoginTimeout 方法——获取连接数据库的等待时间

获取驱动程序试图登录到某一数据库时可以等待的最长时间，以 s 为单位。

方法声明

```
public static int getLoginTimeout()
```

返回值：驱动程序登录时间限制，以 s 为单位。

示例

本示例利用 getLoginTimeout 方法输出了驱动程序登录到某一数据库时可以等待的最长时间为多少秒。

```
System.out.println(DriverManager.getLoginTimeout());
```


PDF电子书基地 http://dayo1982.400gb.com

5.1.6 getLogWriter 方法——获取日志的打印输出流

获取日志 writer。

方法声明

```
public staticPrintWritergetLogWriter()
```

返回值：java.io.PrintWriter 对象。

示例

本示例利用 getLogWriter 方法获取日志 writer，如获取成功则写入日志。

```
PrintWriter out = null;
out = DriverManager.getLogWriter();
if(out != null){
    out.println("得到信息");
    out.close();
}
```

5.1.7 println 方法——输出日志

将一条消息打印到当前 JDBC 日志流中。

方法声明

```
public static void println(String message)
```

入口参数：message 为一条日志或追踪消息。

示例

本示例利用 println 方法将一条消息打印到当前 JDBC 日志流中。

```
DriverManager.println("获取驱动成功！");
```

5.1.8 registerDriver 方法——注册给定的驱动程序

向 DriverManager 注册给定驱动程序。新加载的驱动程序类应该调用 registerDriver 方法让 DriverManager 知道自己。

方法声明

```
public static void registerDriver(Driver driver) throws SQLException
```

入口参数：driver 将向 DriverManager 注册的新的 JDBCDriver。

抛出异常：SQLException。

示例

本示例演示了删除 mysql 驱动进行其它操作后，重新注册了 mysql 驱动。

```
try {
    Class.forName("com.mysql.jdbc.Driver");          //注册 mysql 驱动
    String url = "jdbc:mysql://localhost:3306/test";
    Driver driver = DriverManager.getDriver(url);
```

```
        //删除 mysql 驱动进行其他操作
        DriverManager.deregisterDriver(driver);
        //重新注册 mysql 驱动
        DriverManager.registerDriver(driver);
    } catch(Exception e) {
        e.printStackTrace();
    }
```

5.1.9 setLoginTimeout 方法——设置连接数据库的等待时间

该方法用于设置连接到数据库可以等待的最长时间，以 s 为单位。

方法声明

```
public static void setLoginTimeout(int seconds);
```

入口参数：seconds 是登录时间限制，0 表示没有时间限制。

示例

下面的利用 setLoginTimeout 方法来设置等待连接的最大时间限制为 10s。

```
DriverManager.setLoginTimeout(10);
```

5.1.10 setLogWriter 方法——设置日志的打印输出流

设置由 DriverManager 和所有驱动程序使用的日志/追踪 PrintWriter 对象。

方法声明

```
public static void setLogWriter(PrintWriter out)
```

入口参数：out 为新的日志/追踪 PrintStream 对象；null 表示禁止日志和追踪。

抛出异常：SecurityException，如果安全管理器存在并且其 checkPermission 方法拒绝设置日志 writer。

示例

本示例用将 setLogWriter()方法将日志设置到 C 盘下。

```
try {
     File file = new File("C:\\DriverManager.log");
    PrintWriterout = new PrintWriter(file);
     DriverManager.setLogWriter(out);
} catch(Exception e) {
    e.printStackTrace();
}
```

5.2 Connection 接口——数据库连接接口

Connection 接口位于 java.sql.包，是与特定数据库的连接会话。在连接上下文中执行 sql 语句，并返回结果。

PDF电子书基地 http://dayo1982.400gb.com

5.2.1 常量——Connection 接口的常量

Connection 接口的常量

字　段	说　明	值
TRANSACTION_NONE	指示事物不支持	0
TRANSACTION_READ_UNCOMMITTED	指示不可以发生脏读；不可重复读和虚读可以发生	1
TRANSACTION_READ_COMMITTED	指示可以发生脏读、不可重复读和虚读	2
TRANSACTION_REPEATABLE_READ	指示不可以发生脏读和不可重复读；虚读可以发生	4
TRANSACTION_SERIALIZABLE	指示不可以发生脏读、不可重复读和虚读	8

5.2.2 clearWarnings 方法——清除警告信息

清除为此Connection对象报告的所有警告。调用此方法后，在为此Connection对象报告新的警告前，getWarnings 方法将返回 null。

方法声明

```
void clearWarnings() throws SQLException
```

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 clearWarnings()方法清除所有警告。

```
Connection conn = …                                    //省略部分代码
conn.clearWarnings();                                  //清除所有警告
```

5.2.3 close 方法——关闭数据库连接对象

该方法用于将 Connection 对象关闭，释放此对象的数据库跟 JDBC 资源。

方法声明

```
void close()
        throws SQLException
```

抛出异常：SQLException。

示例

下面的代码利用 close 方法将 Connection 对象关闭。

```
Connection conn = …                                    //省略部分代码
conn.close();                                          //将连接关闭
```

5.2.4 commit 方法——提交事务

该方法用于使当前事务中的更改成为持久的更改，并释放 Connection 对象当

前持有的所有数据库锁。此方法只有在手动事务的模式下才可以用。

方法声明

```
void commit()
        throws SQLException
```

抛出异常：SQLException。

示例

下面的代码利用 commit 方法将当前事务提交。

```
Connection conn = …                                    //省略部分代码
conn.setAutoCommit(false);                             //先将事物设为手动提交
Statement st = conn.createStatement();
String sql = "INSERT INTO users VALUES ('xiaoli','123456')";
st.execute(sql);
conn.commit();                                         //将事物提交
```

5.2.5 createBlob 方法——构造二进制的对象

构造实现 Blob 接口的对象。返回的对象最初不包含任何数据。Blob 接口的 setBinaryStream 和 setBytes 方法可用来向 Blob 添加数据。

方法声明

```
Blob createBlob() throws SQLException
```

返回值：实现 Blob 接口的对象。

抛出异常

SQLException：如果无法构造实现 Blob 接口的对象，在关闭的连接上调用此方法，或者发生数据库访问错误。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此据类型。

示例

本示例调用 Connection 的 createBlob()方法获取了 Blob 接口的对象。

```
Connection conn = …                                    //省略部分代码
Blob blob = conn.createBlob();
```

5.2.6 createClob 方法——构造字符型的对象

构造实现 Clob 接口的对象。返回的对象最初不包含任何数据。Clob 接口的 setAsciiStream、setCharacterStream 和 setString 方法可用来向 Clob 添加数据。

方法声明

```
Clob createClob() throws SQLException
```

返回值：实现 Clob 接口的对象。

抛出异常

SQLException：如果无法构造实现 Clob 接口的对象，在关闭的连接上调用

PDF电子书基地 http://dayo1982.400gb.com

此方法，或者发生数据库访问错误。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此据类型。

示例

本示例调用 Connection 的 createClob()方法获取了 Clob 接口的对象。

```
Connection conn = …                          //省略部分代码
Clob clob = conn.createClob();               //创建 Clob 接口的对象
```

5.2.7 createNClob 方法——构造字节字符的对象

构造实现 NClob 接口的对象。返回的对象最初不包含任何数据。NClob 接口的 setAsciiStream、setCharacterStream 和 setString 方法可用来向 NClob 添加数据。

方法声明

```
NClob createNClob() throws SQLException
```

返回值：实现 NClob 接口的对象。

抛出异常

SQLException：如果无法构造实现 NClob 接口的对象，在关闭的连接上调用此方法，或者发生数据库访问错误。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此据类型。

示例

本示例调用 Connection 的 createNClob()方法获取了 NClob 接口的对象。

```
Connection conn = …                          //省略部分代码
NClob nclob = conn.createNClob();            //创建 NClob 接口的对象
```

5.2.8 createSQLXML 方法——构造 SQLXML 对象

构造实现 SQLXML 接口的对象。返回的对象最初不包含任何数据。SQLXML 接口的 createXmlStreamWriter 和 setString 方法可用来向 SQLXML 对象添加数据。

方法声明

```
SQLXML createSQLXML() throws SQLException
```

返回值：实现 SQLXML 接口的对象。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此据类型。

示例

本示例调用 Connection 的 createNClob()方法获取了 SQLXML 接口的对象。

```
Connection conn = …                                    //省略部分代码
SQLXMLSQLXML= conn.createSQLXML();                     //创建 SQLXML 接口的对象
```

5.2.9 createStatement 方法——创建执行 SQL 语句的对象

该方法用于创建一个 Statement 对象，封装 SQL 语句发送给数据库，通常用来执行不带参数的 SQL 语句。

方法声明 1

```
Statement createStatement()
        throws SQLException
```

返回值：返回 Statement 对象。

抛出异常：SQLException。

示例

获得一个 Statement 对象。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
```

方法声明 2

创建一个 Statement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。此方法与上述 createStatement 方法相同，但它允许重写默认结果集类型和并发性。已创建结果集的可保存性可调用 getHoldability() 确定。

```
Statement createStatement(int resultSetType,
                          int resultSetConcurrency)
                          throws SQLException
```

入口参数

resultSetType 为结果集类型，为以下三种之一。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

resultSetConcurrency 为并发类型为以下两种之一。

- ResultSet.CONCUR_READ_ONLY
- ResultSet.CONCUR_UPDATABLE

返回值：一个新的 Statement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。

抛出异常

SQLException：如果发生数据库访问错误，在关闭的连接上调用此方法，或者给定参数不是指示类型和并发性的 ResultSet 常量。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法，或

PDF电子书基地 http://dayo1982.400gb.com

者对于指定的结果集类型和结果集并发性，此方法不受支持。

示例

本示例获得了一个 Statement 对象，并发类型为只读，结果集类型光标只能向前移动。

```
Connection conn = …                                    //省略部分代码
//创建 Statement 对象
Statement stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                      ResultSet.CONCUR_READ_ONLY);
```

方法声明 3

```
Statement createStatement(int resultSetType,
                          int resultSetConcurrency,
                          int resultSetHoldability)
                          throws SQLException
```

入口参数

resultSetType 为结果集类型，为以下三种之一。

- ResultSet.TYPE_FORWARD_ONL
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

resultSetConcurrency 为并发类型，为以下两种之一。

- ResultSet.CONCUR_READ_ONLY
- ResultSet.CONCUR_UPDATABLE

resultSetHoldability 为以下两种之一。

- ResultSet.HOLD_CURSORS_OVER_COMMIT
- ResultSet.CLOSE_CURSORS_AT_COMMIT

返回值：一个新的 Statement 对象，该对象将生成具有给定类型、并发性和可保存性的 ResultSet 对象。

抛出异常

SQLException：如果发生数据库访问错误，在关闭的连接上调用此方法，或者给定参数不是指示类型和并发性的 ResultSet 常量。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法，或者对于指定的结果集类型和结果集并发性，此方法不受支持。

示例

本示例获得了一个 Statement 对象，并发类型为只读，结果集类型光标只能向前移动，具有此可保存性的打开的 ResultSet 对象将保持开放。

```
Connection conn = …                                    //省略部分代码
//创建 Statement 对象
```

```
Statement stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                      ResultSet.CONCUR_READ_ONLY,
                                      ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

5.2.10 getAutoCommit 方法——获取提交模式

该方法用于获取当前 Connection 对象的提交模式。

方法声明

```
boolean getAutoCommit()
        throws SQLException
```

返回值：Connection 的提交方式如果自动提交，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 getAutoCommit 方法获取 Connection 对象的提交模式。

```
Connection conn = …                                    //省略部分代码
Boolean bool = conn.getAutoCommit();
if(bool) {
    System.out.println("当前为自动提交");
}else{
    System.out.println("当前为手动提交");
}
```

5.2.11 getCatalog 方法——获取当前目录名称

获取此 Connection 对象的当前目录名称。

方法声明

```
String getCatalog() throws SQLException
```

返回值：当前目录名称，如果不存在，则返回 null。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getCatalog()方法输出了 Connection 对象的当前数据库的名称。

```
Connection conn = …                                    //省略部分代码
System.out.println(conn.getCatalog());                 //当前数据库的名称
```

5.2.12 getClientInfo 方法——获取客户端信息列表

返回一个列表，它包含驱动程序支持的每个客户端信息属性的名称和当前值。如果客户端信息属性尚未设置并且没有默认值，那么客户端信息属性的值为

PDF电子书基地 http://dayo1982.400gb.com

null。

方法声明 1

```
Properties getClientInfo() throws SQLException
```

返回值：一个 Properties 对象，它包含驱动程序支持的每个客户端信息属性的名称和当前值。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getClientInfo()方法获取了客户端信息属性集。

```
Connection conn = …                                    //省略部分代码
Properties props = conn.getClientInfo();               //获取客户端信息属性集
```

方法声明 2

返回通过名称指定的客户端信息属性的值。如果指定的客户端信息属性尚未设置并且没有默认值，那么此方法返回 null。如果指定客户端信息属性名称不受驱动程序支持，那么此方法也将返回 null。

```
String getClientInfo(String name) throws SQLException
```

入口参数：name 为要获取的客户端信息属性的名称。

返回值：指定的客户端信息属性的值。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getClientInfo()方法获取了客户端信息 ClientUser 属性的值。

```
Connection conn = …                                    //省略部分代码
//获取 ClientUser 属性的值
String clientUser = conn.getClientInfo("ClientUser");
```

5.2.13 getHoldability 方法——获取 ResultSet 对象的可保存性

获取使用此 Connection 对象创建的 ResultSet 对象的当前可保存性。

方法声明

```
int getHoldability() throws SQLException
```

返回值：可保存性，它是 ResultSet.HOLD_CURSORS_OVER_COMMIT 或 ResultSet.CLOSE_CURSORS_AT_COMMIT 之一。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getHoldability()方法输出了 Connection 对象创建的 ResultSet 对象的当前可保存性的 int 值。

```
Connection conn = …                                    //省略部分代码
System.out.println(conn.getHoldability());             //输出可保存性的值
```

5.2.14 getMetaData 方法——获取数据库的元数据

获取一个 DatabaseMetaData 对象，该对象包含关于此 Connection 对象所连接的数据库的元数据。元数据包括关于数据库的表、受支持的 SQL 语法、存储过程、此连接功能等的信息。

方法声明

```
DatabaseMetaData getMetaData() throws SQLException
```

返回值：此 Connection 对象的一个 DatabaseMetaData 对象。

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getHoldability()方法输出了 Connection 对象创建的 getMetaData 对象创建了一个 DatabaseMetaData 对象。

```
Connection conn = ……                                  //省略部分代码
//获取 DatabaseMetaData 对象
DatabaseMetaData metaData = conn.getMetaData();
```

5.2.15 getTransactionIsolation 方法——获取事务隔离级别

获取此 Connection 对象的当前事务隔离级别。

方法声明

```
int getTransactionIsolation() throws SQLException
```

返回值：当前事务隔离级别，它将是以下常量之一。

Connection.TRANSACTION_READ_UNCOMMITTED

Connection.TRANSACTION_READ_COMMITTED

Connection.TRANSACTION_REPEATABLE_READ

Connection.TRANSACTION_SERIALIZABLE

Connection.TRANSACTION_NONE

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

示例

本示例调用 Connection 的 getTransactionIsolation()方法输出了当前事务的隔离级别。

```
Connection conn = …                                    //省略部分代码
//输出当前事务的隔离级别
System.out.println(conn.getTransactionIsolation());
```

5.2.16 getTypeMap 方法——获取 Map 对象

获取与此 Connection 对象关联的 Map 对象。除非应用程序已经添加了一个条目，否则返回的类型映射将为空。

方法声明

```
Map<String,Class<?>> getTypeMap() throws SQLException
```

返回值：与此 Connection 对象关联的 java.util.Map 对象。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 Connection 的 getTypeMap()方法获取了与此 Connection 对象关联的 Map 对象。

```
Connection conn = …                                    //省略部分代码
Map map = conn.getTypeMap();                           //获取相关联的 Map 对象
```

5.2.17 getWarnings 方法——获取调用报告的第一个警告

获取此 Connection 对象上的调用报告的第一个警告。如果有多个警告，则后续警告将被链接到第一个警告，可以通过对之前获得的警告调用 SQLWarning.getNextWarning 方法获取。

方法声明

```
SQLWarning getWarnings() throws SQLException
```

返回值：第一个 SQLWarning 对象；如果不存在，则返回 null。

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 getWarnings()方法获取了与此 Connection 对象调用报告的第一个警告。

```
Connection conn = …                                    //省略部分代码
SQLWarning warning = conn.getWarnings();               //获取警告
```

5.2.18 isClosed 方法——判断 Connection 对象是否被关闭

该方法用于判断 Connection 对象是否被关闭，如果关闭返回 true，否则返回 false。

方法声明

```
boolean isClosed()
        throws SQLException
```

返回值：如果 Connection 对象已经关闭，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 isClosed 方法检查当前连接是否被关闭。

```
Connection conn = …                                  //省略部分代码
conn.commit();
System.out.println(conn.isClosed());                 //连接没有关闭，输出 false
conn.close();                                        //将连接关闭
System.out.println(conn.isClosed());                 //连接已经被关闭，输出 true
```

5.2.19 isReadOnly 方法——判断 Connection 对象是否为只读模式

该方法用于判断此 Connection 对象是否处于只读模式。

方法声明

```
boolean isReadOnly()
        throws SQLException
```

返回值：返回此连接对象是否是只读模式，如果是则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 isReadOnly 方法判断 Connection 对象是否为只读模式。

```
Connection conn = …                                  //省略部分代码
Boolean bool = conn.isReadOnly();
System.out.println(bool);
```

5.2.20 isValid 方法——判断连接是否有效

如果连接尚未关闭并且仍然有效，则返回 true。驱动程序将提交一个关于该连接的查询，或者使用其他某种能确切验证在调用此方法时连接是否仍然有效的机制。由驱动程序提交的用来验证该连接的查询将在当前事务的上下文中执行。

方法声明

```
boolean isValid(int timeout) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：timeout 是等待用来验证连接是否完成的数据库操作的时间，以 s 为单位。如果在操作完成之前超时期满，则此方法返回 false。0 值表示不对数据库操作应用超时值。

返回值：如果连接有效，则返回 true，否则返回 false。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 isValid()方法输出了 Connection 连接有效。

```
Connection conn = …                                    //省略部分代码
if(conn.isValid(0)){
    System.out.println("Connection 连接有效");
}else{
    System.out.println("Connection 连接无效");
}
```

5.2.21 nativeSQL 方法——返回 SQL 语句的本机形式

将给定的 SQL 语句转换成系统本机 SQL 语法。在发送 JDBCSQL 语法之前，驱动程序可以将其转换成其系统本机 SQL 语法。此方法返回驱动程序将发送的语句的本机形式。

方法声明

```
String nativeSQL(String sql) throws SQLException
```

入口参数：sql 为可以包含一个或多个'?'参数占位符的 SQL 语句。

返回值：此语句的本机形式。

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 nativeSQL()方法输出了此语句的本机形式。

```
Connection conn = ……                                  //省略部分代码
String sql = "select top 10 * from users";
System.out.println(conn.nativeSQL(sql));               //输出 sql 语句的本机形式
```

5.2.22 prepareCall 方法——调用数据库存储过程

创建一个 CallableStatement 对象来调用数据库存储过程。CallableStatement 对象提供了设置其 IN 和 OUT 参数的方法，以及用来执行调用存储过程的方法。

方法声明 1

```
CallableStatement prepareCall(String sql) throws SQLException
```

入口参数：sql 为可以包含一个或多个'?'参数占位符的 SQL 语句。

返回值：包含预编译 SQL 语句的新的默认 CallableStatement 对象。

抛出异常：SQLException，如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 prepareCall()方法执行了一个存储过程 saveuser。

```
Connection conn = …                                    //省略部分代码
String sql = "{call saveuser(?,?)}";                   //调用存储过程
conn.prepareCall(sql);
```

方法声明 2

创建一个 CallableStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。此方法与上述 prepareCall 方法相同，但它允许重写默认结果集类型和并发性。已创建结果集的可保存性可调用 getHoldability() 确定。

```
CallableStatement prepareCall(String sql, int resultSetType,
                              int resultSetConcurrency) throws SQLException
```

入口参数

sql 为可以包含一个或多个'?'参数占位符的 SQL 语句。

resultSetType 为结果集类型，类型如下。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

resultSetConcurrency 并发类型，类型如下。

- ResultSet.CONCUR_READ_ONLY
- ResultSet.CONCUR_UPDATABLE

返回值：一个包含预编译的 SQL 语句的新 CallableStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法，或者对于指定的结果集类型和结果集并发性此方法不受支持。

示例

本示例调用 Connection 的 prepareCall()方法创建一个 CallableStatement 对象重写默认结果集的并发类型为只读，结果集类型光标只能向前移动。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                          //省略部分代码
String sql = "{call saveuser(?,?)}";
CallableStatement cs = conn.prepareCall(sql,
                                ResultSet.TYPE_FORWARD_ONLY,
                                ResultSet.CONCUR_READ_ONLY);
```

方法声明 3

创建一个 CallableStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。

```
CallableStatement prepareCall(String sql, int resultSetType,
        int resultSetConcurrency, int resultSetHoldability)) throws SQLException
```

入口参数

sql 为可以包含一个或多个'?'参数占位符的 SQL 语句。

resultSetType 为结果集类型，类型如下。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

resultSetConcurrency 并发类型，类型如下。

- ResultSet.CONCUR_READ_ONLY
- ResultSet.CONCUR_UPDATABLE

resultSetHoldability 为以下 ResultSet，类型如下。

- ResultSet.HOLD_CURSORS_OVER_COMMIT
- ResultSet.CLOSE_CURSORS_AT_COMMIT

返回值：一个包含预编译 SQL 语句的新 CallableStatement 对象，该对象将生成具有给定类型、并发性和可保存性的 ResultSet 对象。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法，或者对于指定的结果集类型和结果集并发性此方法不受支持。

示例

本示例调用 Connection 的 prepareCall()方法创建一个 CallableStatement 对象重写默认结果集的并发类型为只读，结果集类型光标只能向前移动，可保存性为开。

```
Connection conn = ……                                        //省略部分代码
String sql = "{call saveuser(?,?)}";
CallableStatement cs = conn.prepareCall(sql,
                                ResultSet.TYPE_FORWARD_ONLY,
```

```
                                    ResultSet.CONCUR_READ_ONLY,
                                    ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

5.2.23 prepareStatement 方法——创建一个预处理语句

该方法用于创建一个 PrepareStatement 对象将带有参数的 SQL 语句发送给数据库，也可以发送没有参数的 SQL 语句。又称为预编译指令，有参数或没有参数的 SQL 语句都可以被编译，并存储在 PrepareSataement 对象中，然后可以有效地使用此对象来多次向数据库发送该语句。

方法声明 1

```
PreparedStatement prepareStatement(String sql)
                                    throws SQLException
```

入口参数：sql 是可能包含'?'参数占位符的 SQL 语句。

返回值：包含预编译 SQL 语句的新 PreparedStatement 对象。

抛出异常：SQLException。

示例

下面的代码利用 preparedStatement 方法获得一个 PreparedStatement 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM tablename WHERE userId = ?";
PreparedStatement ps = conn.prepareStatement(sql);
```

方法声明 2

```
PreparedStatement prepareStatement(String sql,
                                    int autoGeneratedKeys)
                                    throws SQLException
```

入口参数

sql 是可能包含'?'参数占位符的 SQL 语句。

autoGeneratedKeys 是 Statement 接口定义的常量。

返回值：能返回带有自动生成主键的 PreparedStatement 对象。

抛出异常：SQLException。

示例

下面的代码利用 preparedStatement 方法获取一个能获得自动生成主键的 PreparedStatement 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM tablename WHERE userId = ?";
PreparedStatement ps = conn.prepareStatement(sql,
                                    Statement.RETURN_GENERATED_KEYS);
```


PDF电子书基地 http://dayo1982.400gb.com



5.2.24 rollback 方法——事务回滚

该方法用于取消在当前事务中进行的更改，并释放当前 Connection 对象当前持有的所有数据库锁。此方法只有在手动事务模式下才可以用。

方法声明 1

```
void rollback()
        throws SQLException
```

抛出异常：SQLException。

示例

下面的代码利用 rollback 方法将对记录的删除操作撤销。

```
Connection conn = ……                                          //省略部分代码
conn.setAutoCommit(false);                                    //先将事物设为手动提交
Statement st = conn.createStatement();
String sql = "DELETE FROM users WHERE username = 'xiaoli'";
st.execute(sql);
conn.roolback();                                              //将事物回滚
```

方法声明 2

取消所有设置给定 Savepoint 对象之后进行的更改。此方法只应该在已禁用自动提交时使用。

```
void rollback(Savepoint savepoint) throws SQLException
```

入口参数：savepoint 要回滚到的 Savepoint 对象。

抛出异常：

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 Connection 的 roolback()方法取消所有设置给定 Savepoint 对象之后进行的更改。

```
Connection conn = …                                           //省略部分代码
conn.setAutoCommit(false);                                    //先将事物设为手动提交
Statement st = conn.createStatement();
String sql = "DELETE FROM users WHERE username = 'xiaoli'";
st.execute(sql);
conn.roolback(spName);                                        //将事物回滚到保存点
```

5.2.25 setAutoCommit 方法——设置数据库连接的提交模式

该方法用于设置数据库连接的提交模式。如果是自动模式，所有的 SQL 语句被执行，并作为单个事物被提交，否则 SQL 语句会被聚集到事物当中，直到调用 commit()方法或 rollback()方法为止。默认情况下新建连接为自动提交模式。

方法声明

```
void setAutoCommit(Boolean autoCommit)
                    throws SQLException
```

入口参数：autoCommit 是设置连接对象的提交方式。

抛出异常：SQLException。

示例

下面的代码利用 setAutoCommit 方法将 Connection 对象设为手动提交方式。

```
Connection conn = …                                    //省略部分代码
conn.setAutoCommit(false);
```

5.2.26 setCatalog 方法——设置目录名称

设置给定目录名称，以便选择要在其中进行工作的此 Connection 对象数据库的子空间。

方法声明

```
void setCatalog(String catalog) throws SQLException
```

入口参数：catalog 在其中进行工作的目录（此 Connection 对象的数据库的子空间）的名称。

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 setCatalog()方法设置为其他的数据库。

```
Connection conn = …                                    //省略部分代码
conn.setCatalog("mingri"); //设置为其他的数据库
```

5.2.27 setClientInfo 方法——设置客户端信息

设置连接的客户端信息属性的值。Properties 对象包含将要设置的客户端属性的名称和值。包含在属性列表中的客户端信息属性集将替换连接上的当前客户端信息属性集。如果当前在连接上设置的某个属性不在属性列表中，则清除该属性。指定一个空属性列表将清除连接上的所有属性。

方法声明 1

```
void setClientInfo(Properties properties) throws SQLClientInfoException
```

入口参数：properties 为要设置的客户端信息属性列表。

抛出异常：SQLClientInfoException，如果在数据库服务器上设置 clientInfo 值时数据库服务器返回一个错误，或者在关闭的连接上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例调用 Connection 的 setClientInfo()方法设置了连接的客户端信息属性的值。

```
Connection conn = ……                                    //省略部分代码
Properties props = new Properties();
props.setProperty("ClientUser", "guest");
props.setProperty("ClientHostname", "mr");
conn.setClientInfo(props); //设置连接的客户端信息属性的值
```

方法声明 2

将 name 指定的客户端信息属性的值设置为 value 指定的值。

```
void setClientInfo(String name, String value) throws SQLClientInfoException
```

入口参数

name：要设置的客户端信息属性名称。

value：要设置为客户端信息属性的值。如果该值为 null，则清除指定属性的当前值。

抛出异常：SQLClientInfoException，如果在数据库服务器上设置 clientInfo 值时数据库服务器返回一个错误，或者在关闭的连接上调用此方法。

示例

本示例调用 Connection 的 setClientInfo()方法设置了连接的客户端信息属性的值。

```
Connection conn = …                                      //省略部分代码
conn.setClientInfo("ClientUser", "guest");               //设置连接的客户端信息属性的值
```

5.2.28 setHoldability 方法——设置 ResultSet 对象的可保存性

将使用此 Connection 对象创建的 ResultSet 对象的默认可保存性（holdability）更改为给定可保存性。ResultSet 对象的默认可保存性可调用 DatabaseMetaData.getResultSetHoldability()确定。

方法声明

```
void setHoldability(int holdability) throws SQLException
```

入口参数：holdabilityResultSet 的可保存性常量，它是 ResultSet.HOLD_CURSORS_OVER_COMMIT 或 ResultSet.CLOSE_CURSORS_AT_COMMIT。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的连接上调用此方法。

SQLFeatureNotSupportedException：如果给定可保存性不受支持。

示例

本示例调用 Connection 的 setHoldability()方法在它创建的 ResultSet 对象提交当前事务时，具有此可保存性的打开的 ResultSet 对象将保持开放。

```
Connection conn = …                                    //省略部分代码
//更改为给定可保存性
conn.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

5.2.29 setReadOnly 方法——设置连接只读模式

该方法用于将此连接设为只读模式提高性能。此方法不能在事务处理期间调用。

方法声明

```
void setReadOnly(boolean readOnly)
                    throws SQLException
```

入口参数：readOnly 是设置连接对象是否使用只读方式。

抛出异常：SQLException。

示例

下面的代码利用 setReadOnly 方法将数据库连接对象设为只读。

```
Connection conn = ……                                   //省略部分代码
conn.setReadOnly(true);
```

5.2.30 setSavepoint 方法——创建一个未命名的保存点

在当前事务中创建一个未命名的保存点（savepoint），并返回表示它的新 Savepoint 对象。如果在活动事务范围之外调用 setSavepoint，则将在新创建的保存点上启动事务。

方法声明 1

```
Savepoint setSavepoint() throws SQLException
```

返回值：新的 Savepoint 对象。

抛出异常

SQLException：如果发生数据库访问错误，在参与分布式事务的同时调用此方法，在关闭的连接上调用此方法，或者此 Connection 对象当前处于自动提交模式。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 Connection 的 setSavepoint()方法创建了一个未命名的保存点。

```
Connection conn = ……                                   //省略部分代码
conn.setSavepoint();                                   //创建一个未命名的保存点
```


PDF电子书基地 http://dayo1982.400gb.com



方法声明 2

该方法用于使用指定的名称在当前事务中创建一个保存点，并返回一个新的 Savepoint 对象。

```
Savepoint setSavepoint(String name)
                    throws SQLException
```

入口参数：name 是保存点名称。

返回值：返回 Savepoint 对象。

抛出异常：SQLException。

示例

下面的代码利用 setSavepoint 方法创建一个保存点。

```
Connection conn = …                                    //省略部分代码
Savepoint sp = conn.setSavepoint("spName");
```

5.2.31 releaseSavepoint 方法——移除指定的 SavePoint 实例

该方法用于从当前事务中移除指定的 SavePoint 实例。

方法声明

```
void releaseSavepoint(Savepoint savepoint)
                    throws SQLException
```

入口参数：savepoint 是将要被移除的 Savepoint 对象。

抛出异常：SQLException。

示例

下面的代码利用 releaseSavepoint 方法移除一个保存点。

```
Connection conn = …                                    //省略部分代码
conn.releaseSavepoint(spName);                         //spName 为保存过的保存点对象
```

5.2.32 setTransactionIsolation 方法——设置事务隔离级别

该方法用于将 Connection 对象的事务隔离级别设置为指定的值，可以设置的事务级别是在 Connection 接口中定义的常量。

方法声明

```
void setTransactionIsolation(int level)
                    throws SQLException
```

入口参数：level 是在 Connection 中定义的常量。

抛出异常：SQLException。

示例

下面的代码利用 setTransactionIsolation 方法将 Connection 对象的事务隔离级别设为不可以发生脏读、不可重复读和虚读。

```
Connection conn = …                                    //省略部分代码
int level = TRANSACTION_SERIALIZABLE;
conn.setTransactionIsolation(level);
```

注意

不能使用 Connection.TRAN SACTION_NONE，因为它指定了不受支持的事务。

5.3 Statement 接口——执行 SQL 语句接口

Statement 接口位于 java.sql.包中，用于执行指定的 SQL 语句，并返回执行结果。默认情况下一个 Statement 对象只能打开一个 ResultSet 对象。

5.3.1 常量——Statement 接口的常量

Statement 接口的常量表

字　段	说　明	值
CLOSE_ALL_RESULTS	调用 getMoreResults 时应该关闭以前一直打开的所有 ResultSet 对象	3
CLOSE_CURRENT_RESULT	调用 getMoreResults 时应该关闭当前 ResultSet 对象	1
EXECUTE_FAILED	在执行批量语句时发生错误	-3
KEEP_CURRENT_RESULT	调用 getMoreResults 时应该关闭当前 ResultSet 对象	2
NO_GENERATED_KEYS	指示生成的键应该不可用于获取	2
RETURN_GENERATED_KEYS	指示生成的键应该可用于获取	1
SUCCESS_NO_INFO	批量语句执行成功但不存在受影响的可用行数计数	-2

5.3.2 addBatch 方法——添加批处理语句

将给定的 SQL 命令添加到此 Statement 对象的当前命令列表中。通过调用方法 executeBatch 可以批量执行此列表中的命令。

方法声明

```
void addBatch(String sql) throws SQLException
```

入口参数：sql 为通常此参数为 SQL INSERT 或 UPDATE 语句。

抛出异常

SQLException：如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者驱动程序不支持批量更新。

示例

本示例通过 Statement 的 addBatch()方法添加了 3 条 sql 语句，然后批量执行。

```
Connection conn = ……                                  //省略部分代码
String sql1 = "insert into users(uname) values('aaa')";
String sql2 = "insert into users(uname) values('bbb')";
String sql3 = "insert into users(uname) values('ccc')";
```


PDF电子书基地 http://dayo1982.400gb.com

```
Statement stmt = conn.createStatement();
stmt.addBatch(sql1);                                    //添加 sql 语句
stmt.addBatch(sql2);
stmt.addBatch(sql3);
stmt.executeBatch();                                    //批量执行
stmt.close();
```

5.3.3 cancel 方法——中止 SQL 语句

如果 DBMS 和驱动程序都支持中止 SQL 语句，则取消此 Statement 对象。一个线程可以使用此方法取消另一个线程正在执行的语句。

方法声明

```
void cancel() throws SQLException
```

抛出异常

SQLException：如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 Statement 的 cancel()方法中止了 SQL 语句。

```
Connection conn = …                                     //省略部分代码
Statement stmt = conn.createStatement();
stmt.cancel();
```

5.3.4 clearBatch 方法——清除批处理语句

清空此 Statement 对象的当前 SQL 命令列表。

方法声明

```
void clearBatch() throws SQLException
```

抛出异常：SQLException，如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者驱动程序不支持批量更新。

示例

本示例通过 Statement 的 clearBatch()方法清空了当前 SQL 命令列表。

```
Connection conn = …                                     //省略部分代码
Statement stmt = conn.createStatement();
String sql1 = "insert into users(uname) values('aaa')";
stmt.addBatch(sql1);                                    //添加 sql 语句
stmt.clearBatch();
```

5.3.5 clearWarnings 方法——清除所有警告

清除在此 Statement 对象上报告的所有警告。调用此方法后，在为此 Statement 对象报告新的警告之前，getWarnings 方法将一直返回 null。

方法声明

```
void clearWarnings() throws SQLException
```

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的Statement 上调用此方法。

示例

本示例通过 Statement 的 clearBatch()方法清空了当前 SQL 命令列表。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();
stmt. clearWarnings();
```

5.3.6 execute 方法——执行 SQL 语句

该方法用于执行任意指定的 SQL 语句。

方法声明

```
boolean execute(String sql)
        throws SQLException
```

入口参数：sql 是任何 SQL 语句。

返回值：判断第一个结果是否为 ResultSet 对象如果是，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 execute 方法删除一条记录。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "DELETE FROM users WHERE username = 'xiaoli' ";
Boolean bool;
bool = st.execute(sql);
System.out.println(bool);                              //没有返回结果集所以打印 false
```

示例

下面的代码利用 execute 方法查询记录。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "SELECT * FROM users";
Boolean bool;
bool = st.execute();
System.out.println(bool);                              //返回 true
```

5.3.7 executeBatch 方法——执行批处理语句

将一批命令提交给数据库来执行，如果全部命令执行成功，则返回更新计数组成的数组。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
int[] executeBatch() throws SQLException
```

返回值：包含批中每个命令的一个元素的更新计数所组成的数组。数组的元素根据将命令添加到批中的顺序排序。

抛出异常：SQLException，如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者驱动程序不支持批量语句。如果未能正确执行发送到数据库的命令之一或者尝试返回结果集合，则抛出 BatchUpdateException（SQLException 的子类）。

示例

本示例通过 Statement 的 executeBatch()方法批量执行了 sql 命令，并返回了包含批中每个命令的一个元素的更新计数所组成的数组。

```
Connection conn = …                                          //省略部分代码
Statement stmt = conn.createStatement();
int[] is = null;
String sql1 = "insert into users(uname) values('aaa')";
String sql2 = "update users set uname = 'xxx' where uname = 'aaa'";
String sql3 = "insert into users(uname) values('ccc')";
stmt.addBatch(sql1);                                         //添加 sql 语句
stmt.addBatch(sql2);
stmt.addBatch(sql3);
is = stmt.executeBatch();                                    //批量执行
```

5.3.8 executeUpdate 方法——执行更新操作

该方法用于执行不返回任何内容的 SQL 语句，如 INSERT、UPDATE、DELETE。

方法声明 1

```
int executeUpdate(String sql)
                  throws SQLException
```

入口参数：不返回内容的 SQL 语句。

返回值：返回受影响的行数合计。

抛出异常：SQLException。

示例

下面的代码利用 executeUpdate 方法获得删除掉的行数合计。

```
Connection conn = …                                          //省略部分代码
Statement st = conn.createStatement();
String sql = "DELETE FROM users";
int count;
count = st.executeUpdate(sql);
System.out.println(count);                                   //打印删除掉的行数合计
```

方法声明 2

```
int executeUpdate(String sql,
                  int autoGeneratedKeys)
                      throws SQLException
```

入口参数

sql 是不返回内容的 SQL 语句。

autoGeneratedKeys 是否可以获取自动生成主键的标志。

返回值：返回受影响的行数合计。

抛出异常：SQLException。

示例

下面的代码利用 executeUpdate 方法将自动生成的主键设置为可用于获取。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "INSERT INTO users VALUES ('xiaoli','123456')";
int i = Statement.RETURN_GENERATED_KEYS;
st.executeUpdate(sql,i);
```

5.3.9 executeQuery 方法——执行查询操作

该方法用于执行指定的 SQL 查询语句，返回一个 Resultset 对象。

方法声明

```
ResultSet executeQuery(String sql)
                      throws SQLException
```

入口参数：要发送给数据库的 SELETE 语句。

返回值：ResultSet 对象。

抛出异常：SQLException。

示例

下面的代码利用 executeQuery 方法获得 ResultSet 对象。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "SELECT * FROM users";
ResultSet rs;
rs = st.executeQuery(sql);
```

5.3.10 getConnection 方法——获取数据库连接对象

该方法用于获取生成此 Statement 对象的数据库连接对象。

方法声明

```
Connection getConnection()
                  throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：返回生成此对象的 Connection。

抛出异常：SQLException。

示例

下面的代码利用 getConnection 方法获得一个 Connection 对象。

```
Connection conn1 = …                                    //省略部分代码
Statement st = conn1.createStatement();
Connection conn2 = st.getConnection();                  //声明一个 Connection 对象
```

5.3.11 getFetchDirection 方法——获取从数据库表获取行的方向

获取从数据库表获取行的方向，该方向是根据此 Statement 对象生成的结果集合的默认值。如果此 Statement 对象没有通过调用方法 setFetchDirection 设置获取方向，则返回值是特定于实现的。

方法声明

```
int getFetchDirection() throws SQLException
```

返回值：根据此 Statement 对象生成的结果集合的默认获取方向。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getFetchDirection()方法获取从数据库表获取行的方向。

```
Connection conn = …                                     //省略部分代码
Statement stmt = conn.createStatement();                //获取 Statement
System.out.println(stmt.getFetchDirection());
```

5.3.12 getFetchSize 方法——获取结果集的行数

获取结果集的行数，该数是根据此 Statement 对象生成的 ResultSet 对象的默认获取大小。如果此 Statement 对象没有通过调用方法 setFetchSize 设置获取大小，则返回值是特定于实现的。

方法声明

```
int getFetchSize() throws SQLException
```

返回值：根据此 Statement 对象生成的结果集合的默认获取大小。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getFetchSize()方法获了 Statement 对象生成的

ResultSet 对象的默认获取大小。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt. getFetchSize());
```

5.3.13 getGeneratedKeys 方法——获取自动生成的键

获取由于执行此 Statement 对象而创建的所有自动生成的键。如果此 Statement 对象没有生成任何键，则返回空的 ResultSet 对象。

方法声明

```
ResultSet getGeneratedKeys() throws SQLException
```

返回值：包含通过执行此 Statement 对象自动生成的键的 ResultSet 对象。

抛出异常：

SQLException：如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

如果未指定表示自动生成键的列，则 JDBC 驱动程序实现将确定最能表示自动生成键的列。

示例

本示例通过 Statement 的 getGeneratedKeys()方法获取执行此 Statement 对象而创建的所有自动生成的键。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
ResultSet rs = stmt.getGeneratedKeys();
```

5.3.14 getMaxFieldSize 方法——获取最大字节数

获取可以为此 Statement 对象所生成 ResultSet 对象中的字符和二进制列值返回的最大字节数。此限制仅应用于 BINARY、VARBINARY、LONGVARBINARY、CHAR、VARCHAR、NCHAR、NVARCHAR、LONGNVARCHAR 和 LONGVARCHAR 列。如果超过了该限制，则直接丢弃多出的数据。

方法声明

```
int getMaxFieldSize() throws SQLException
```

返回值：存储字符和二进制值的列的当前列大小限制；0 表示没有任何限制。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例通过 Statement 的 getGeneratedKeys()方法输出了所生成 ResultSet 对象中的字符和二进制列值返回的最大字节数。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt.getMaxFieldSize());
```

5.3.15 getMaxRows 方法——获取最大行数

获取由此 Statement 对象生成的 ResultSet 对象可以包含的最大行数。如果超过了此限制，则直接撤销多出的行。

方法声明

```
int getMaxRows() throws SQLException
```

返回值：由此 Statement 对象生成的 ResultSet 对象的当前最大行数；0 表示不存在任何限制。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getMaxRows()方法输出了生成的 ResultSet 对象可以包含的最大行数。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt. getMaxRows());
```

5.3.16 getMoreResults 方法——移动到 Statement 对象的下一个结果

移动到此 Statement 对象的下一个结果，如果其为 ResultSet 对象，则返回 true，并隐式关闭利用方法 getResultSet 获取的所有当前 ResultSet 对象。

方法声明 1

```
boolean getMoreResults() throws SQLException
```

返回值：如果下一个结果为 ResultSet 对象，则返回 true；如果其为更新计数或不存在更多结果，则返回 false。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getMoreResults()方法判断下一结果是否为 ResultSet 对象。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();
if(stmt.getMoreResults()){
    System.out.println("下一个结果为 ResultSet 对象");
}
```

方法声明 2

将此 Statement 对象移动到下一个结果，根据给定标志指定的指令处理所有当前 ResultSet 对象；如果下一个结果为 ResultSet 对象，则返回 true。

```
boolean getMoreResults(int current) throws SQLException
```

入口参数： current 下列 Statement 常量之一，这些常量指示将对使用 getResultSet 方法获取的当前 ResultSet 对象发生的操作，如下所示。

- Statement.CLOSE_CURRENT_RESULT
- Statement.KEEP_CURRENT_RESULT
- Statement.CLOSE_ALL_RESULTS

返回值：如果下一个结果为 ResultSet 对象，则返回 true；如果其为更新计数或不存在更多结果，则返回 false。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getMoreResults()方法判断下一结果是否为 ResultSet 对象。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();
if(stmt.getMoreResults(Statement.CLOSE_ALL_RESULTS)){
    System.out.println("下一个结果为 ResultSet 对象");
}
```

5.3.17 getQueryTimeout 方法——获取等待执行的秒数

获取驱动程序等待 Statement 对象执行的秒数。如果超过该限制，则抛出 SQLException。

方法声明

```
int getQueryTimeout() throws SQLException
```

返回值：以 s 为单位的当前查询超时限制，0 表示不存在任何限制。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getMaxRows()方法输出了获取驱动程序等待 Statement 对象执行的秒数。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt. getQueryTimeout());
```

5.3.18 getResultSet 方法——获取结果集

以 ResultSet 对象的形式获取当前结果。每个结果只应调用一次此方法。

方法声明

```
ResultSet getResultSet() throws SQLException
```

返回值：以 ResultSet 对象的形式返回当前结果；如果结果是更新计数或没有更多的结果，则返回 null。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getResultSet()方法获取了 ResultSet 对象，并输出了其中的值。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
String sql = "select * from users";
stmt.executeQuery(sql);
ResultSet rs = stmt.getResultSet();
while(rs.next()){
    System.out.println(rs.getInt("id"));
}
```

5.3.19 getResultSetConcurrency 方法——获取结果集并发性

获取此 Statement 对象生成的 ResultSet 对象的结果集并发性。

方法声明

```
int getResultSetConcurrency() throws SQLException
```

返回值：ResultSet.CONCUR_READ_ONLY 或 ResultSet.CONCUR_UPDATABLE。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getResultSetConcurrency()方法输出了生成的 ResultSet 对象的结果集并发性的值。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt.getResultSetConcurrency());
```

5.3.20 getResultSetHoldability 方法——获取结果集可保存性

获取此 Statement 对象生成的 ResultSet 对象的结果集可保存性。

方法声明

```
int getResultSetHoldability() throws SQLException
```

返回值：ResultSet.HOLD_CURSORS_OVER_COMMIT 或 ResultSet.CLOSE_CURSORS_AT_COMMIT。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getResultSetHoldability()方法输出了生成的 ResultSet 对象的结果集可保存性。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt. getResultSetHoldability());
```

5.3.21 getResultSetType 方法——获取结果集类型

获取此 Statement 对象生成的 ResultSet 对象的结果集类型。

方法声明

```
int getResultSetType() throws SQLException
```

返回值：ResultSet.TYPE_FORWARD_ONLY 或 ResultSet.TYPE_SCROLL_INSENSITIVE 或 ResultSet.TYPE_SCROLL_SENSITIVE。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getResultSetType()方法输出了生成的 ResultSet 对象的结果集类型。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt. getResultSetType());
```

5.3.22 getUpdateCount 方法——获取更新的记录数

以更新计数的形式获取当前结果；如果结果为 ResultSet 对象或没有更多结果，则返回-1。每个结果只应调用一次此方法。

方法声明

```
int getUpdateCount() throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：以更新计数的形式返回当前结果；如果当前结果为 ResultSet 对象或没有更多结果，则返回-1。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getUpdateCount()方法输出了执行 sql 语句更新的记录数。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
String sql = "update users set uname = 'xxx' where uname = 'aaa'";
stmt.executeUpdate(sql2);                              //执行 sql 语句
System.out.println(stmt.getUpdateCount());             //输出更新的行数
```

5.3.23 getWarnings 方法——获取调用报告的第一个警告

获取此 Statement 对象上的调用报告的第一个警告。后续的 Statement 对象警告将链接到此 SQLWarning 对象上。每次（重新）执行语句时，都会自动清除警告链。

方法声明

```
SQLWarning getWarnings() throws SQLException
```

返回值：第一个 SQLWarning 对象；如果不存在任何警告，则返回 null。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 getWarnings()方法获取了它调用报告的第一个警告。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
SQLWarning war = stmt.getWarnings();
```

5.3.24 setMaxRows 方法——获取最大行数

该方法用于设置此 Statement 对象生成的所有 ResultSet 对象的最大行数，如果超出了最大限制，直接撤销超出的部分。

方法声明

```
void setMaxRows(int max)
                    throws SQLException
```

入口参数：最大行限制，0 为不限制。

抛出异常：SQLException。

示例

下面代码利用 setMaxRows 方法设置当前 Statement 对象生成的 ResultSet 的最大行数限制为 20 行。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
st.setMaxRows(20);
```

5.3.25 getMaxRows 方法——获取最大行数限制值

该方法用于获得当前 Statement 对象生成的 ResultSet 对象的最大行数限制值。

方法声明

```
int getMaxRows()
        throws SQLException
```

返回值：最大行数，0 表示不存在限制。

抛出异常：SQLException。

示例

获得当前 Statement 对象生成的 ResultSet 对象的最大行数限制值。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
System.out.println(st.getMaxRows());                   //在没有设置的时候打印 0
```

5.3.26 close 方法——关闭 Statement 对象

该方法用于释放此 Statement 对象的数据库和 JDBC 资源。关闭 Statement 对象时还将关闭当前的 ResultSet 对象。

方法声明

```
void close()
        throws SQLException
```

抛出异常：SQLException。

示例

下面代码利用 close 方法将当前的 Statement 对象关闭。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "SELECT * FROM users";
st.executeQuery(sql);
st.close();
```

5.3.27 isClosed 方法——判断 Statement 对象是否关闭

该方法用于检查当前的 Statement 对象是否关闭。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
boolean isClosed()
        throws SQLException
```

返回值：如果 Statement 对象已经关闭，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面代码利用 isClosed 方法检查 Statement 是否已经关闭。

```
Connection conn = …                                    //省略部分代码
Statement st = conn.createStatement();
String sql = "SELECT * FROM users";
st.executeQuery(sql);
System.out.println(st.isClosed());                     //没有关闭时打印 false
st.close();                                            //将 Statement 对象关闭
System.out.println(st.isClosed());                     //关闭以后打印 true
```

5.3.28 isPoolable 方法——判断对象是否可池化

返回指示 Statement 对象是否是可池化的值。

方法声明

```
boolean isPoolable() throws SQLException
```

返回值：如果 Statement 是可池化的，则返回 true，否则返回 false。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 isPoolable()方法获取并输出 Statement 是否可池化。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
System.out.println(stmt.isPoolable());                 //输出 Statement 是否可池化
```

5.3.29 setCursorName 方法——设置指针名称

将 SQL 指针名称设置为给定的 String，后续 Statement 对象的 execute 方法将使用此字符串。然后，可以在 SQL 位置更新或删除语句中使用该名称来标识由此语句生成的 ResultSet 对象中的当前行。如果数据库不支持位置更新/删除，则此方法为无操作(noop)。为了确保指针具有支持更新的适当隔离级别，指针的 SELECT 语句的形式应该为 SELECT FOR UPDATE。如果 FOR UPDATE 不存在，则位置更新可能失败。

方法声明

```
void setCursorName(String name) throws SQLException
```

入口参数：name 为新指针名称，该名称在连接中必须是唯一的。

抛出异常

SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

SQLFeatureNotSupportedException 如果 JDBC 驱动程序不支持此方法。

注意

按照定义，位置更新和删除必须由生成的用于放置的 ResultSet 对象之外的某个 Statement 对象来执行。此外，在连接中指针名称还必须是唯一的。

示例

本示例通过 Statement 的 setCursorName()方法将指针名称设置为 pname。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
stmt.setCursorName("pname");                           //指针名称设置为 pname
```

5.3.30 setEscapeProcessing 方法——设置是否进行转义处理

将转义处理设置为开或关。如果转义扫描为开启（默认值），则驱动程序在将 SQL 语句发送到数据库之前执行转义替换。

方法声明

```
void setEscapeProcessing(boolean enable) throws SQLException
```

入口参数：enable 为 true 表示启用转义处理，为 false 表示禁用转义处理。

抛出异常：SQLException 如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法。

注意

因为准备的语句通常在进行此调用之前解析，所以对 PreparedStatements 对象禁用转义处理无效。

示例

本示例通过 Statement 的 setEscapeProcessing()方法将转义扫描设置为开启，驱动程序在将 SQL 语句发送到数据库之前执行转义替换。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
stmt.setEscapeProcessing(true);                        //转义扫描设置为开启
```

5.3.31 setFetchDirection 方法——设置处理行的方向

向驱动程序提供关于方向的提示，在使用此 Statement 对象创建的 ResultSet 对象中将按该方向处理行。默认值为 ResultSet.FETCH_FORWARD。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
void setFetchDirection(int direction) throws SQLException
```

入口参数：direction 用于处理行的初始方向。

抛出异常：SQLException，如果发生数据库访问错误，或者在已关闭的 Statement 上调用此方法，或者给定的方向不是 ResultSet.FETCH_FORWARD、ResultSet.FETCH_REVERSE 和 ResultSet.FETCH_UNKNOWN 之一。

注意

此方法为此 Statement 对象生成的结果集合设置默认获取方向。每个结果集合都具有它自己用于获取和设置其自身获取方向的方法。

示例

本示例通过 Statement 的 setFetchDirection()方法对象创建的 ResultSet 对象中将按正方向处理行。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
//按正向处理结果集中的行
stmt.setFetchDirection(ResultSet.FETCH_FORWARD);
```

5.3.32 setFetchSize 方法——设置结果集合的最大行数

为 JDBC 驱动程序提供一个提示，它提示此 Statement 生成的 ResultSet 对象需要更多行时应该从数据库获取的行数。指定的行数仅影响使用此语句创建的结果集合。如果指定的值为 0，则忽略该提示。默认值为 0。

方法声明

```
void setFetchSize(int rows) throwsSQLException
```

入口参数：rows 为要获取的行数。

抛出异常：SQLException 如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者不满足条件 rows≥0。

示例

本示例通过 Statement 的 setFetchSize()方法设置此 Statement 生成的 ResultSet 对象需要更多行时应该从数据库获取 50 行。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
stmt.setFetchSize(50);
```

5.3.33 setMaxFieldSize 方法——设置可返回的最大字节数

设置此 Statement 对象生成的 ResultSet 对象中字符和二进制列值可以返回的最大字节数限制。此限制仅应用于 BINARY、VARBINARY、LONGVARBINARY、

CHAR、VARCHAR、NCHAR、NVARCHAR、LONGNVARCHAR 和 LONGVARCHAR 字段。如果超过了该限制，则直接丢弃多出的数据。为了获得最大的可移植性，应该使用大于 256 的值。

方法声明

```
void setMaxFieldSize(int max) throwsSQLException
```

入口参数：max 为以字节为单位的新列大小限制，0 表示没有任何限制。

抛出异常：SQLException 如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者不满足条件 max≥0。

示例

本示例通过 Statement 的 setFetchSize()方法设置了 Statement 生成的 ResultSet 对象中字符和二进制列值可以返回的最大字节数为 1000。

```
Connection conn = ……                                    //省略部分代码
Statement stmt = conn.createStatement();                 //获取 Statement
stmt.setMaxFieldSize(1000);
```

5.3.34 setMaxRows 方法——设置最大行数

将此 Statement 对象生成的所有 ResultSet 对象可以包含的最大行数限制设置为给定数。如果超过了该限制，则直接撤销多出的行。

方法声明

```
void setMaxRows(int max) throws SQLException
```

入口参数：max 为以字节为单位的新列大小限制，0 表示没有任何限制。

抛出异常：SQLException 如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者不满足条件 max≥0。

示例

本示例通过 Statement 的 setMaxRows()方法设置了 Statement 生成的 ResultSet 对象可以包含的最大行数为 20。

```
Connection conn = ……                                    //省略部分代码
Statement stmt = conn.createStatement();                 //获取 Statement
stmt. setMaxRows(20);
```

5.3.35 setPoolable 方法——设置语句是否可池化

请求将 Statement 池化或非池化。指定值是语句池实现的一个提示，指示应用程序是否希望语句被池化。关于是否使用该提示，由语句池管理器决定。

语句的可池化的值对驱动程序实现的内部语句缓存以及应用程序服务器和其他应用程序实现的外部语句缓存都适用。默认情况下，Statement 在创建时不是可池化的，而 PreparedStatement 和 CallableStatement 在创建时是可池化的。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
void setPoolable(boolean poolable) throws SQLException
```

入口参数：poolable 如果为 true，则请求将语句池化；如果为 false，则不池化语句。

抛出异常：SQLException，如果在已关闭的 Statement 上调用此方法。

示例

本示例通过 Statement 的 setPoolable()方法设置了请求将语句池化。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
stmt.setPoolable(true);                                //请求将语句池化
```

5.3.36 setQueryTimeout 方法——设置等待执行的秒数

将驱动程序等待 Statement 对象执行的秒数设置为给定秒数。如果超过该限制，则抛出 SQLException。

方法声明

```
void setQueryTimeout(int seconds) throws SQLException
```

入口参数：seconds 以 s 为单位的查询超时限制，0 表示没有任何限制。

抛出异常：SQLException 如果发生数据库访问错误，在已关闭的 Statement 上调用此方法，或者不满足条件 seconds≥0。

示例

本示例通过 Statement 的 setQueryTimeout()方法将驱动程序等待 Statement 对象执行的秒数设置为 5s。

```
Connection conn = …                                    //省略部分代码
Statement stmt = conn.createStatement();               //获取 Statement
stmt. setQueryTimeout(5);
```

5.4 PreparedStatement 接口——预编译 SQL 语句接口

PreparedStatement 接口位于 java.sql 包，它继承自 Statement 接口。表示预编译的 SQL 语句对象。SQL 语句被预先编译并存储在 PreparedStatement 对象中，然后可以使用该对象多次高效的执行该语句。

5.4.1 clearParameters 方法——立即清除当前参数值

立即清除当前参数值，通常参数值对语句的重复使用仍然有效。设置一个参数值会自动清除其以前的值。不过，在某些情况下，直接释放当前参数值使用的资源也是很有用的；这可以通过调用 clearParameters 方法实现。

方法声明

```
void clearParameters() throws SQLException
```

抛出异常：SQLException 如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

示例

本示例通过 PreparedStatement 的 clearParameters()方法清除当前参数值，所以执行时抛出异常。

```
Connection conn = …                                    //省略部分代码
String sql = "update users set uname = ? where uname = ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, "aaa");
ps.setString(2, "xxx");
ps.clearParameters();
ps.executeUpdate();
```

5.4.2 execute 方法——执行 SQL 语句

该方法用于执行任意指定的 SQL 语句，并判断第一个结果是否为 ResultSet 对象。

方法声明

```
boolean execute()
        throws SQLException
```

返回值：判断第一个结果是否为 ResultSet 对象如果是，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面代码利用 execute 方法查询记录。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
PreparedStatement ps = conn.prepareStatement(sql);
boolean bool = ps.execute();
System.out.println(bool);                              //有结果集被返回，打印 true
```

5.4.3 executeQuery 方法——执行查询语句

该方法用于执行 SQL 查询语句，并返回该语句生成的 ResultSet 对象。

方法声明

```
ResultSet executeQuery()
                throws SQLException
```

返回值：ResultSet 对象。

PDF电子书基地 http://dayo1982.400gb.com

抛出异常：SQLException。

示例

下面的代码利用 executeQuery 方法获得一个 ResultSet 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
```

5.4.4 executeUpdate 方法——执行更新语句

该方法用于在此 PreparedStatement 对象中执行 SQL 语句，该语句必须是一个数据操作语句，比如 INSERT、UPDATE、DELETE，或者是无返回内容的 SQL 语句。

方法声明

```
int executeUpdate()
                throws SQLException
```

返回值：数据操作语句返回受影响的行数，无返回内容的 SQL 返回 0。

抛出异常：SQLException。

示例

下面的代码利用 executeUpdate 方法向数据库插入一条记录。

```
Connection conn = …                                    //省略部分代码
String sql = "INSERT INTO users(username,pwd) VALUES ("lisi","123456")";
PreparedStatement ps = conn.prepareStatement(sql);
int i =ps.executeUpdate();                             //执行
System.out.println(i);                                 //成功插入记录，打印 1
```

5.4.5 getMetaData 方法——获取 ResultSet 对象列的信息

获取包含有关 ResultSet 对象列信息的 ResultSetMetaData 对象，ResultSet 对象将在执行此 PreparedStatement 对象时返回。

因为 PreparedStatement 对象被预编译，所以不必执行就可以知道它将返回的 ResultSet 对象。因此，可以对 PreparedStatement 对象调用 getMetaData 方法，而不必等待执行该对象，然后再对返回的 ResultSet 对象调用 ResultSet.getMetaData 方法。

方法声明

```
ResultSetMetaData getMetaData() throws SQLException
```

返回值：ResultSet 对象列的描述；如果驱动程序无法返回一个 ResultSetMetaData 对象，则返回 null。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement

上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 PreparedStatement 的 getMetaData()方法获取了 ResultSet 对象列的描述。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSetMetaData rsmd = ps.getMetaData();
System.out.println(rsmd.getColumnName(1));
```

5.4.6 getParameterMetaData 方法——获取参数信息

获取此 PreparedStatement 对象的参数的编号、类型和属性。

方法声明

```
ParameterMetaData getParameterMetaData() throws SQLException
```

返回值：一个 ParameterMetaData 对象，它包含有关此 PreparedStatement 对象的每个参数标记的编号、类型和属性的信息。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

示例

本示例通过 PreparedStatement 的 getParameterMetaData()方法获取了它的 ParameterMetaData。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ParameterMetaData pmd = ps.getParameterMetaData();
```

5.4.7 setAsciiStream 方法——将指定参数设置为给定输入流

将指定参数设置为给定输入流。在将一个非常大的 ASCII 值输入到 LONGVARCHAR 参数时，通过 java.io.InputStream 发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。JDBC 驱动程序将执行从 ASCII 到数据库 char 格式的任何必要转换。

方法声明 1

```
void setAsciiStream(int parameterIndex, InputStream x)
    throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，……

x：包含 ASCII 参数值的 java 输入流。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过PreparedStatement的setAsciiStream()方法将输入流中的内容写入数据库。

```
Connection conn = …                                    //省略部分代码
InputStream is = new FileInputStream(new File("c:\\1.txt"));
String sql = "insert into users(uname) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setAsciiStream(1, is);
ps.executeUpdate();
is.close();
ps.close();
```

方法声明 2

将指定参数设置为给定输入流，该输入流将具有给定字节数。在将一个非常大的 ASCII 值输入 LONGVARCHAR 参数时，通过 java.io.InputStream 发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。JDBC 驱动程序将执行从 ASCII 到数据库 char 格式的任何必要转换。

```
void setAsciiStream(int parameterIndex, InputStream x, int length)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

x：包含 ASCII 参数值的 java 输入流。

Length：流中的字节数。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过PreparedStatement的setAsciiStream()方法将输入流中的内容写入到数据库（写入4个字节）。

```
Connection conn = ……                                    //省略部分代码
InputStream is = new FileInputStream(new File("c:\\1.txt"));
String sql = "insert into users(uname) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setAsciiStream(1, is, 4);
ps.executeUpdate();
is.close();
ps.close();
```

方法声明 3

将指定参数设置为给定输入流，该输入流将具有指定字节数。在将一个非常大的 ASCII 值输入到 LONGVARCHAR 参数时，通过 java.io.InputStream 发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。JDBC 驱动程序将执行从 ASCII 到数据库 char 格式的任何必要转换。

```
void setAsciiStream(int parameterIndex, InputStream x, long length)
    throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

x：包含 ASCII 参数值的 java 输入流。

length：流中的字节数。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过PreparedStatement的setAsciiStream()方法将输入流中的内容写入到数据库（写入4个字节）。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                    //省略部分代码
InputStream is = new FileInputStream(new File("c:\\1.txt"));
String sql = "insert into users(uname) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setAsciiStream(1, is, 4);
ps.executeUpdate();
is.close();
ps.close();
```

5.4.8 setBigDecimal 方法——设置大数字参数值

将指定参数设置为给定的 java.math.BigDecimal 值。在将此值发送到数据库时，驱动程序将它转换成一个 SQL NUMERIC 值。

方法声明

```
void setBigDecimal(int parameterIndex, BigDecimal x)
                    throws SQLException
```

入口参数：parameterIndex，第一个参数是 1，第二个参数是 2，…

x 为参数值。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 PreparedStatement 的 setBigDecimal()方法将 BigDecimal 值写入数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(uname,age) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, "abc");
ps.setBigDecimal(2,new BigDecimal(123));
ps.executeUpdate();
ps.close();
```

5.4.9 setBinaryStream 方法——设置二进制参数值

将指定参数设置为给定输入流。在将一个非常大的二进制值输入 LONGVARBINARY 参数时，通过 java.io.InputStream 对象发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。

方法声明 1

```
void setBinaryStream(int parameterIndex, InputStream x) throws SQLException
```

入口参数：parameterIndex，第一个参数是1，第二个参数是2，…

x为包含二进制参数值的java输入流。

抛出异常

SQLException：如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误，或者在关闭的PreparedStatement上调用此方法。

SQLFeatureNotSupportedException：如果JDBC驱动程序不支持此方法。

注意

查询JDBC驱动程序文档，以确定使用带length参数的setBinaryStream是否更有效。

示例

本示例通过PreparedStatement的setBinaryStream()方法将二进制输入流值写入到数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(id,uname,photo) values(?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 1);
ps.setString(2, "明日科技");
InputStream is = new FileInputStream("c:\\22.jpg");
ps.setBinaryStream(3, is);
ps.executeUpdate();
ps.close();
is.close();
```

方法声明2

将指定参数设置为给定输入流，该输入流将具有给定字节数。在将一个非常大的二进制值输入LONGVARBINARY参数时，通过java.io.InputStream对象发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。

```
void setBinaryStream(int parameterIndex, InputStream x, int length)
    throws SQLException
```

入口参数

parameterIndex：第一个参数是1，第二个参数是2，…

x：包含二进制参数值的java输入流。

Length：流中的字节数。

抛出异常

SQLException：如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误，或者在关闭的PreparedStatement上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setBinaryStream()方法将二进制输入流值写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(id,uname,photo) values(?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 1);
ps.setString(2, "明日科技");
InputStream is = new FileInputStream("c:\\22.jpg");
int length = is.available();
ps.setBinaryStream(3, is, length);
ps.executeUpdate();
ps.close();
is.close();
```

方法声明 3

将指定参数设置为给定输入流，该输入流将具有指定字节数。在将一个非常大的二进制值输入 LONGVARBINARY 参数时，通过 java.io.InputStream 对象发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。

```
void setBinaryStream(int parameterIndex, InputStream x, int length)
    throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，……

x：包含二进制参数值的 java 输入流。

length：流中的字节数。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setBinaryStream()方法将二进制输入流值写入数据库。

```
Connection conn = ……                                    //省略部分代码
String sql = "insert into users(id,uname,photo) values(?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 1);
ps.setString(2, "明日科技");
InputStream is = new FileInputStream("c:\\22.jpg");
long length = is.available();
ps.setBinaryStream(3, is, length);
ps.executeUpdate();
ps.close();
is.close();
```

5.4.10 setBlob 方法——设置 Blob 型参数

将指定参数设置为给定 java.sql.Blob 对象。在将此对象发送到数据库时，驱动程序将它转换成一个 SQL BLOB 值。

方法声明 1

```
void setBlob(int parameterIndex, Blob x) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，……

x：一个映射 SQL BLOB 值的 Blob 对象。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 PreparedStatement 的 setBlob()方法将 BLOB 对象写入到数据库。

```
Connection conn = …                                     //省略部分代码
String sql = "insert into users(id,userdata) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 4);
InputStream is = new FileInputStream("c:\\编程词典.rar");
byte[] b = new byte[is.available()];
is.read(b);
Blob blob = new SerialBlob(b);
ps.setBlob(2, blob);
ps.executeUpdate();
ps.close();
is.close();
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

将指定参数设置为 InputStream 对象。此方法不同于 setBinaryStream（int, InputStream）方法，因为它会通知驱动程序应该将参数值作为 Blob 发送给服务器。在使用 setBinaryStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGVARBINARY 还是 Blob 发送给服务器。

```
void setBinaryStream(int parameterIndex, InputStream x, int length)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，……

inputStream：包含用于设置参数值的数据的对象。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

查询 JDBC 驱动程序文档，以确定使用带 length 参数的 setBlob 是否更有效。

示例

本示例通过 PreparedStatement 的 setBlob()方法将输入流对象写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(id,userdata) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 4);
InputStream is = new FileInputStream("c:\\编程词典.rar");
ps.setBlob(2, is);
ps.executeUpdate();
ps.close();
is.close();
```

方法声明 3

将指定参数设置为 InputStream 对象。Inputstream 必须包含 length 指定的字符数，否则在执行 PreparedStatement 时将生成一个 SQLException。此方法不同于 setBinaryStream（int, InputStream, int）方法，因为它会通知驱动程序应该将参数值作为 Blob 发送给服务器。在使用 setBinaryStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGVARBINARY 还是 Blob 发送给服务器。

```
void setBlob(int parameterIndex, InputStream inputStream, long length)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

inputStream：包含用于设置参数值的数据的对象。

length：参数数据中的字节数。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 PreparedStatement 的 setBlob()方法将输入流对象写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(id,userdata) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, 4);
InputStream is = new FileInputStream("c:\\编程词典.rar");
long length = is.available();
ps.setBlob(2, is, length);
ps.executeUpdate();
ps.close();
is.close();
```

5.4.11 setBoolean 方法——设置布尔型参数

该方法用于将指定的参数设置为给定 Java boolean 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL BIT 或 boolean 值。

方法声明

```
void setBoolean(int parameterIndex,
                boolean x)
                    throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面代码利用 setBoolean 方法给参数赋值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM users WHERE isPass = ?";
PreparedStatement ps = conn.prepareStatement(sql);
Boolean bool = true;
```


PDF电子书基地 http://dayo1982.400gb.com

```
ps.setBoolean(1, bool);                                    //进行赋值
ps.executeQuery();
```

5.4.12 setByte 方法——设置字节型参数

该方法用于将指定的参数设置为给定的 Java byte 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL TINYINT 值。

方法声明

```
void setByte(int parameterIndex,
             byte x)
             throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setByte 方法给参数赋值。

```
Connection conn = …                                        //省略部分代码
String sql = "SELECT * FROM users WHERE isPass = ?";
PreparedStatement ps = conn.prepareStatement(sql);
byte b = 1;
ps.setByte(1,b);
ps.executeQuery();
```

5.4.13 setBytes 方法——设置字节数组型参数

该方法用于将指定的参数设置为给定的 Java byte 数组。此值被传送到数据库时，驱动程序会将它转换成一个 SQL VARBINARY 或 LONGVARBINARY 值。

方法声明

```
void setBytes(int parameterIndex,
              byte[] x)
                  throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setBytes 方法给参数赋值。

```
Connection conn = …
//省略部分代码
```

```
String sql = "UPDATE myTable set myData = ? where id = 1";
//构建一个输入流
FileInputStream in = new FileInputStream("c:\\a.txt");
//声明一个 byte 类型数组，长度为要储存的输入流长度
byte[] b = new byte[in.available()];
//将输入流读入到 byte 数组中
in.read(b);
PreparedStatement ps = conn.prepareStatement(sql);
ps.setBytes(1,b);                                          //为参数赋值
ps.execute();
```

5.4.14 setCharacterStream 方法——将指定参数设置为给定 Reader 对象

将指定参数设置为给定 Reader 对象。在将一个非常大的 UNICODE 值输入 LONGVARCHAR 参数时，通过 java.io.Reader 对象发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。JDBC 驱动程序将执行从 UNICODE 到数据库 char 格式的任何必要转换。

方法声明 1

```
void setCharacterStream(int parameterIndex, Reader reader)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含 Unicode 数据的 java.io.Reader 对象。

抛出异常：SQLException，如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setCharacterStream()方法将字符流内容写入到数据库。

```
Connection conn = …                                        //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setCharacterStream(1, r);
ps.executeUpdate();
r.close();
ps.close();
ps.close();
```


PDF电子书基地 http://dayo1982.400gb.com



方法声明 2

将给定参数设置为给定 Reader 对象，该对象具有给定字符数长度。在将一个非常大的 UNICODE 值输入 LONGVARCHAR 参数时，通过 java.io.Reader 对象发送它可能更为实际。将根据需要从流中读取数据，一直读取到文件末尾。JDBC 驱动程序将执行从 UNICODE 到数据库 char 格式的任何必要转换。

```
void setCharacterStream(int parameterIndex, Reader reader, int length)
    throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含 Unicode 数据的 java.io.Reader 对象。

length：流中的字符数。

抛出异常：SQLException 如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setCharacterStream()方法将字符流内容写入数据库，写入了 100 个字符。

```
Connection conn = ……                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setCharacterStream(1, r, 100);
ps.executeUpdate();
r.close();
ps.close();
ps.close();
```

5.4.15 setClob 方法——设置 Clob 型参数

将指定参数设置为给定 java.sql.Clob 对象。在将此对象发送到数据库时，驱动程序将它转换成一个 SQL Clob 值。

方法声明 1

```
void setClob(int parameterIndex, Clob x) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

x：一个映射 SQL Clob 值的 Clob 对象。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setClob()方法将字 Clob 对象映射为 SQL Clob 值写入数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
char[] cs = "i like java".toCharArray();
Clob clob = new SerialClob(cs);
ps.setClob(1, clob);
ps.executeUpdate();
ps.close();
```

方法声明 2

将指定参数设置为 Reader 对象。此方法不同于 setCharacterStream（int, Reader）方法，因为它会通知驱动程序应该将参数值作为 Clob 发送给服务器。在使用 setCharacterStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGVARCHAR 还是 Clob 发送给服务器。

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含用于设置参数值的数据的对象。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

查询 JDBC 驱动程序文档，以确定使用带 length 参数的 setClob 是否更有效。

示例

本示例通过 PreparedStatement 的 setClob()方法将字符流作为 CLOB 写入数

PDF电子书基地 http://dayo1982.400gb.com

据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setClob(1, r);
ps.executeUpdate();
r.close();
ps.close();
```

方法声明 3

将指定参数设置为 Reader 对象。Reader 必须包含 length 指定的字符数，否则在执行 PreparedStatement 时将生成一个 SQLException。此方法不同于 setCharacterStream（int, Reader, int）方法，因为它会通知驱动程序应该将参数值作为 Clob 发送给服务器。在使用 setCharacterStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGVARCHAR 还是 Clob 发送给服务器。

```
void setClob(int parameterIndex, Reader reader, long length)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含 Unicode 数据的 java.io.Reader 对象。

length：流中的字符数。

抛出异常：SQLException，如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

示例

本示例通过 PreparedStatement 的 setClob()方法将字符流作为 CLOB 写入数据库，写入了 100 个字符。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setClob(1, r, 100);
ps.executeUpdate();
r.close();
ps.close();
```

5.4.16 setDate 方法——设置日期型参数

该方法用于将指定的参数设置为给定的 java.sql.Date 数组。此值被传送到数据库时，驱动程序会将它转换成一个 SQL DATE。

方法声明

```
void setDate(int parameterIndex,
             Date x)
             throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setDate 方法为参数赋值。

```
Connection conn = …                                          //省略部分代码
//构建一个 java.sql.Date 对象
Date date = new Date(new java.util.Date().getTime());
String sql = "INSERT INTO users(username,pwd,regDate) VALUES (?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1,"zhangsan");
ps.setString(2,"123456");
ps.setDate(3,date);                                          //给参数赋值
ps.execute();
```

5.4.17 setDouble 方法——设置双精度型参数

该方法用于将指定的参数设置为给定的 java double 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL DOUBLE 值。

方法声明

```
void setDouble(int parameterIndex,
               double x)
               throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setDouble 方法为参数赋值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM user WHERE number = ?";
```


PDF电子书基地 http://dayo1982.400gb.com

```
PreparedStatement ps = conn.prepareStatement(sql);
double d = 59.88;
ps.setDouble(1, d);
```

5.4.18 setFloat 方法——设置单精度型参数

该方法用于将指定的参数设置为给定的 java REAL 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL FLOAT 值。

方法声明

```
void setFloat(int parameterIndex,
              float x)
                  throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setFloat 方法为参数赋值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM user WHERE number = ?";
PreparedStatement ps = conn.prepareStatement(sql);
float f = 98.56f;
ps.setFloat(1,f);
```

5.4.19 setInt 方法——设置整型参数

该方法用于将指定的参数设置为给定的 java int 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL INTEGER 值。

方法声明

```
void setInt(int parameterIndex,
            int x)
                throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setInt 方法为参数赋值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM user WHERE number = ?";
```

```
PreparedStatement ps = conn.prepareStatement(sql);
int i = 23;
ps.setInt(1,i);
```

5.4.20　setLong 方法——设置长整型参数

该方法用于将指定的参数设置为给定的 java long 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL BIGINT 值。

方法声明

```
void setLong(int parameterIndex,
             long x)
                throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setLong 方法为参数赋值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM user WHERE number = ?";
PreparedStatement ps = conn.prepareStatement(sql);
long l = 999999;
ps.setLong(1, l);
```

5.4.21　setNCharacterStream 方法——将指定参数设置为 Reader 对象

将指定参数设置为 Reader 对象。Reader 将读取数据，直到到达文件末尾。驱动程序执行从 Java 字符格式到数据库中国家字符集的必要转换。

方法声明 1

```
void setNCharacterStream(int parameterIndex, Reader value)
      throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

value：参数值。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自

PDF电子书基地 http://dayo1982.400gb.com

己的子类。

示例

本示例通过 PreparedStatement 的 setNCharacterStream()方法将输入流中的内容写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setNCharacterStream(1, r);
ps.executeUpdate();
r.close();
ps.close();
```

方法声明 2

将指定参数设置为 Reader 对象。Reader 将读取数据，直到到达指定的字符数。驱动程序执行从 java 字符格式到数据库字符集的必要转换。

```
void setNCharacterStream(int parameterIndex, Reader value, long length)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

x：包含 ASCII 参数值的 java 输入流。

length：参数数据中的字符数。

抛出异常

SQLException：如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setNCharacterStream()方法将输入流中的内容写入到数据库，写入了 100 个字符。

```
Connection conn = ……                                        //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setNCharacterStream(1, r, 100);
```

```
ps.executeUpdate();
r.close();
ps.close();
```

5.4.22 setNClob 方法——设置 NClob 类型的参数

将指定参数设置为 java.sql.NClob 对象。在将此对象发送到数据库时，驱动程序将它转换成一个 SQL NCLOB 值。

方法声明 1

```
void setNClob(int parameterIndex, NClob value) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

value：参数值。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

此流对象既可以是一个标准 java 流对象，也可以是实现标准接口的用户自己的子类。

示例

本示例通过 PreparedStatement 的 setNClob()方法将字 NClob 对象映射为 SQL Clob 值写入到数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
NClob nc = conn.createNClob();
nc.setString(1, "明日科技");
nc.setString(5, "编程词典");
ps.setNClob(1, nc);
ps.executeUpdate();
ps.close();
```

方法声明 2

将指定参数设置为 Reader 对象。此方法不同于 setCharacterStream（int, Reader）方法，因为它会通知驱动程序应该将参数值作为 NCLOB 发送给服务器。在使用 setCharacterStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGNVARCHAR 还是 NCLOB 发送给服务器。

```
void setNClob(int parameterIndex, Reader reader) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含用于设置参数值的数据的对象。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

查询 JDBC 驱动程序文档，以确定使用带 length 参数的 setNClob 是否更有效。

示例

本示例通过 PreparedStatement 的 setNClob()方法将字符流作为 NCLOB 写入到数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setNClob(1, r);
ps.executeUpdate();
r.close();
ps.close();
```

方法声明 3

将指定参数设置为 Reader 对象。Reader 必须包含 length 指定的字符数，否则在执行 PreparedStatement 时将生成一个 SQLException。此方法不同于 setCharacterStream（int, Reader, int）方法，因为它会通知驱动程序应该将参数值作为 NCLOB 发送给服务器。在使用 setCharacterStream 方法时，驱动程序可能必须做一些额外的工作，以确定应该将参数数据作为 LONGNVARCHAR 还是 NCLOB 发送给服务器。

```
void setNClob(int parameterIndex, Reader reader, long length)
    throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

reader：包含 Unicode 数据的 java.io.Reader 对象。

length：流中的字符数。

抛出异常：SQLException 如果 parameterIndex 不对应于 SQL 语句中的参数

标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

示例

本示例通过 PreparedStatement 的 setNClob()方法将字符流作为 NCLOB 写入到数据库，写入了 100 个字符。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
File file = new File("C:\\desc.txt");
Reader r = new FileReader(file);
ps.setNClob(1, r, 100);
ps.executeUpdate();
r.close();
ps.close();
```

5.4.23 setNString 方法——设置 String 类型的参数

将指定参数设置为给定 String 对象。在将此对象发送给数据库时，驱动程序将它转换成一个 SQLNCHAR、NVARCHAR 或 LONGNVARCHAR 值（取决于该参数相对于驱动程序在 NVARCHAR 值上的限制的大小）。

方法声明

```
void setNString(int parameterIndex, String value) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

value：参数值。

抛出异常

SQLException：如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果驱动程序不支持国家字符（national character）集；如果驱动程序检测到可能发生数据转换错误；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例通过 PreparedStatement 的 setNString()方法将 String 值写入数据库。驱动程序会自动将它转换成一个 NCHAR、NVARCHAR 或 LONGNVARCHAR 值。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(description) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
String s = "明日科技编程词典";
ps.setNString(1, s);
ps.executeUpdate();
ps.close();
```


PDF电子书基地 http://dayo1982.400gb.com

5.4.24 setShort 方法——设置 Short 类型的参数

该方法用于将指定的参数设置为给定的 java short 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL SMALLINT 值。

方法声明

```
void setShort(int parameterIndex,
              short x)
              throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setShort 方法为参数赋值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM user WHERE number = ?";
PreparedStatement ps = conn.prepareStatement(sql);
short s =225;
ps.setShort(1,s);
```

5.4.25 setString 方法——设置 String 类型的参数

该方法用于指定参数设置为给定的 Java String 值。在将此值发送给数据库时，驱动程序将它转换成一个 SQL VARCHAR。

方法声明

```
void setString(int parameterIndex,
               String x)
               throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面代码利用 setString 方法为参数赋值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM user WHERE username = ?";
PreparedStatement ps = conn.prepareStatement(sql);
String name ="zhangsan";
ps.setString(1,name);
```

5.4.26 setTime 方法——设置时间类型的参数

该方法用于将指定的参数设置为给定的 Java time 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL TIME 值。

方法声明 1

```
void setTime(int parameterIndex,
    Time x)
      throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setTime 方法为参数赋值。

```
Connection conn = …                                    //省略部分代码
//构建一个 java.sql.Time 对象
Time time = new Time(new Date().getTime());
String sql = "INSERT INTO users(username,pwd,regTime) VALUES (?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1,"zhangsan");
ps.setString(2,"123456");
ps.setDate(3,time);                                    //给参数赋值
ps.execute();
```

方法声明 2

使用给定的 Calendar 对象将指定参数设置为给定 java.sql.Time 值。驱动程序构造一个 SQL TIME 值，该值随后被驱动程序发送给数据库。利用 Calendar 对象，驱动程序可以在考虑自定义时区的情况下计算时间。如果没有指定任何 Calendar 对象，那么驱动程序将使用默认时区，该时区是运行应用程序的虚拟机所在的时区。

```
void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException
```

入口参数

parameterIndex：第一个参数是 1，第二个参数是 2，…

x：参数值。

cal：驱动程序将用来构造日期的 Calendar 对象。

抛出异常：SQLException，如果 parameterIndex 不对应于 SQL 语句中的参数标记；如果发生数据库访问错误，或者在关闭的 PreparedStatement 上调用此方法。

注意

此方法需要数据库和它的驱动支持。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例通过 PreparedStatement 的 setTime()方法使用使用日本时区将 Time 值写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(id,birth) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
//获取当前时间
Time time = new Time(new java.util.Date().getTime());
//构造一个时区的 Calendar
Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("Japan"));
ps.setInt(1, 2);
ps.setTime(2, time, cal);
ps.executeUpdate();
ps.close();
```

5.4.27 setTimestamp 方法——设置参数值为时间戳

该方法用于将指定的参数设置为给定的 java.sql.Timestamp 值。此值被传送到数据库时，驱动程序会将它转换成一个 SQL TIMESTAMP 值。

方法声明 1

```
void setTimestamp(int parameterIndex,
                  Timestamp x)
                  throws SQLException
```

入口参数

parameterIndex：对应参数的序号。

x：参数的值。

抛出异常：SQLException。

示例

下面的代码利用 setTimestamp 方法给参数赋值。

```
Connection conn = …                                          //省略部分代码
//构建一个 java.sql.Timestamp 对象
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
Date date = sdf.parse("2008-08-08 20:08:08");
Timestamp ts = new Timestamp(date.getTime());
String sql = "SELECT * FROM myTable WHERE startdate = ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setTimestamp(1,ts);                                       //给参数赋值
ps.executeQuery();
```

方法声明 2

将指定参数设置为给定 java.sql.Timestamp 值。在将此值发送到数据库时，驱动程序将它转换成一个 SQL TIMESTAMP 值。

```
void setTimestamp(int parameterIndex, Timestamp x, Calendar cal)
        throws SQLException
```

入口参数

parameterIndex：第一个参数是1，第二个参数是2，…

x：参数值。

cal：驱动程序将用来构造日期的Calendar对象。

抛出异常：SQLException，如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误，或者在关闭的PreparedStatement上调用此方法。

注意

此方法需要数据库和它的驱动支持。

示例

本示例通过PreparedStatement的setTimestamp()方法使用使用日本时区将Timestamp值写入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(id,birth) values(?,?)";
PreparedStatement ps = conn.prepareStatement(sql);
//获取当前时间
Timestamp t = new Timestamp(new java.util.Date().getTime());
//构造一个时区的 Calendar
Calendar c = Calendar.getInstance(TimeZone.getTimeZone("Japan"));
ps.setInt(1, 2);
ps.setTimestamp(2, t, c);
ps.executeUpdate();
ps.close();
```

5.4.28 setURL方法——设置参数为URL

将指定参数设置为给定java.net.URL值。在将此值发送到数据库时，驱动程序将它转换成一个SQL DATALINK值。

方法声明

```
void setURL(int parameterIndex, URL x) throws SQLException
```

入口参数

parameterIndex：第一个参数是1，第二个参数是2，…

x：要设置的java.net.URL对象。

抛出异常

SQLException：如果parameterIndex不对应于SQL语句中的参数标记；如果发生数据库访问错误，或者在关闭的PreparedStatement上调用此方法。

SQLFeatureNotSupportedException：如果JDBC驱动程序不支持此方法。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例通过 PreparedStatement 的 setURL()方法将 URL 值插入到数据库。

```
Connection conn = …                                          //省略部分代码
String sql = "insert into users(url) values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
URL url = new URL("http://www.mingrisoft.com/");
ps.setURL(1, url);
ps.executeUpdate();
ps.close();
```

5.5 ResultSet 接口——结果集接口

ResultSet 接口位于 java.sql 包中，表示数据库的结果集，通常通过执行查询数据库的语句生成。

5.5.1 常量——ResultSet 接口的常量

ResultSet 接口的常量

字　段	说　明	值
CLOSE_CURSORS_AT_COMMIT	提交当前事务时，具有此可保存性的打开的 ResultSet 对象将被关闭	2
CONCUR_READ_ONLY	不可以更新的 ResultSet 对象的并发模式	1 007
CONCUR_UPDATABLE	可以更新的 ResultSet 对象的并发模式	1 008
FETCH_FORWARD	按正向处理结果集中的行	1 000
FETCH_REVERSE	按逆向处理结果集中的行	1 001
FETCH_UNKNOWN	结果集中的行的处理顺序未知	1 002
HOLD_CURSORS_OVER_COMMIT	提交当前事务时，具有此可保存性的打开的 ResultSet 对象将保持开放	1
TYPE_FORWARD_ONLY	该常量指示光标只能向前移动的 ResultSet 对象的类型	1 003
TYPE_SCROLL_INSENSITIVE	该常量指示可滚动但通常不受 ResultSet 底层数据更改影响的 ResultSet 对象的类型	1 004
TYPE_SCROLL_SENSITIVE	该常量指示可滚动并且通常受 ResultSet 底层数据更改影响的 ResultSet 对象的类型	1 005

5.5.2 absolute 方法——将光标移动到指定行

将光标移动到此 ResultSet 对象的给定行编号。如果行编号为正，从第一行开始移动，如果为负，从最后一行开始移动。

方法声明

```
boolean absolute(int row) throws SQLException
```

入口参数：row 光标应该移动到的行的编号。正的编号指示从结果集开头开

始计数的行编号；负的编号指示从结果集末尾开始计数的行编号。

返回值：如果光标移动到此 ResultSet 对象的位置处，则返回 true；如果光标在第一行的前面或最后一行的后面，则返回 false。

抛出异常：SQLException。

示例

本示例利用 absolute 方法将光标移到 Request 对象第 5 行。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
boolean b = rs.absolute(5);                            //移动光标
```

示例

本示例利用 absolute 方法将光标移到 Request 对象倒数第 5 行。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
boolean b = rs.absolute(-5);                           //移动光标
```

5.5.3 afterLast 方法——将光标移动到最后一行

该方法用于将光标移动到 Request 对象的最后一行之后，如果结果集中不包含任何行，此方法无效。

方法声明

```
void afterLast()
        throws SQLException
```

抛出异常：SQLException。

示例

下面代码利用 afterLast 方法将光标移到 Request 对象最后一行之后。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
rs.afterLast();                                        //移动光标
```

5.5.4 beforeFirst 方法——将光标移动到第一行之前

将光标移动到 Request 对象的第一行之前，如果结果集中不包含任何行，此方法无效。

方法声明

```
void beforeFirst()
        throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

抛出异常：SQLException。

示例

下面代码利用 beforeFirst 方法将光标移动到第一行之前。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
rs.beforeFirst();                                      //移动光标
```

5.5.5 cancelRowUpdates 方法——取消对当前行所作的更新

取消对 ResultSet 对象中的当前行所作的更新。此方法在调用更新方法之后，但在调用 updateRow 方法之前调用才可以回滚对行所作的更新。如果没有进行任何更新或者已经调用 updateRow 方法，则此方法无效。

方法声明

```
void cancelRowUpdates() throws SQLException
```

抛出异常

SQLException：如果发生数据库访问错误；在已关闭的结果集上调用此方法；结果集并发性为 CONCUR_READ_ONLY，或者在光标不位于插入行上时调用了此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例利用 cancelRowUpdates 方法取消更新了 ResultSet 对象第 5 行数据。

```
Connection conn = ……                                  //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //光标移动到第 5 行
    if(rs.absolute(5)){
        rs.updateString("uname", "abc");               //设置更新值
        rs.cancelRowUpdates();                         //取消更新
        rs.updateRow();    //更新数据
    }
}
rs.close();
ps.close();
```

5.5.6 clearWarnings 方法——清除所有警告

清除在此 ResultSet 对象上报告的所有警告。调用此方法后，在为此 ResultSet

对象报告新的警告之前，getWarnings 方法将返回 null。

方法声明

```
void clearWarnings() throws SQLException
```

抛出异常：SQLException，如果发生数据库访问错误；在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 clearWarnings 方法清除了警告信息。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
rs.clearWarnings();
rs.close();
ps.close();
```

5.5.7 close 方法——关闭 ResultSet 对象

该方法用于关闭 ResultSet 对象，立即释放此对象的数据库跟 JDBC 资源。

方法声明

```
void close()
    throws SQLException
```

抛出异常：SQLException。

示例

下面的代码利用 close 方法将 ResultSet 对象关闭。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
rs.close();                                            //关闭 ResultSet 对象
```

5.5.8 deleteRow 方法——删除当前行

从此 ResultSet 对象和底层数据库中删除当前行。光标不位于插入行上时不能调用此方法。

方法声明

```
void deleteRow() throws SQLException
```

抛出异常

SQLException：如果发生数据库访问错误；在已关闭的结果集上调用此方法；结果集并发性为 CONCUR_READ_ONLY，或者在光标不位于插入行上时调用了此方法。

PDF电子书基地 http://dayo1982.400gb.com

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 deleteRow 方法删除了第 1 行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
//获取更新的 ResultSet 对象
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
        rs.deleteRow();                                      //删除此行
}
rs.close();
ps.close();
```

5.5.9 findColumn 方法——获取 Resultset 的列索引

将给定的 ResultSet 列标签映射到其 ResultSet 列索引。

方法声明

```
int findColumn(String columnLabel) throws SQLException
```

入口参数：columnIndex 使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：给定列名称的列索引。

抛出异常

SQLException：如果发生数据库访问错误；在已关闭的结果集上调用此方法；结果集并发性为 CONCUR_READ_ONLY，或者在光标不位于插入行上时调用了此方法。

示例

本示例调用了 ResultSet 对象的 findColumn 方法输出了字段 uname 所在的列的索引。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    int i = rs.findColumn("uname");
    System.out.println("uname 所在列为：" + i);
}
rs.close();
ps.close();
```

5.5.10 first 方法——将光标移动到第一行

该方法用于将光标移动到 ResultSet 对象的第一行。

方法声明

```
boolean first()
        throws SQLException
```

返回值：移动是否成功，如果成功，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面代码利用 first 方法将光标移动到第一行。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
Boolean bool = rs.first();                             //移动光标
System.out.println(bool);                              //打印移动结果
```

5.5.11 getAsciiStream 方法——获取指定列的 ASCII 字符流的值

以 ASCII 字符流的形式获取此 ResultSet 对象的当前行中指定列的值。然后，可以按块从流中读取值。此方法尤其适合于获取很大的 LONGVARCHAR 值。JDBC 驱动程序将执行从数据库格式到 ASCII 的任何必要转换。

方法声明 1

```
InputStream getAsciiStream(int columnIndex) throws SQLException
```

入口参数：columnIndex 第一个列是 1，第二个列是 2，…

返回值：以一字节 ASCII 字符流的形式返回传递数据库列值的 java 输入流；如果值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

注意

在获取任何其他列的值之前必须读取返回流中的所有数据。下一次调用获取方法将隐式关闭该流。此外，当调用 InputStream.available 方法时，不管是否存在可用数据，流都可能返回 0。

示例

本示例调用了 ResultSet 对象的 getAsciiStrea 方法以 ASCII 字符流的形式获取字段值，并输出到控制台。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = rs.getAsciiStream(5);
    OutputStream os = System.out;
    int i;
    while(( i = is.read()) != -1){
        os.write(i);
    }
    is.close();
    os.flush();
    os.close();
}
rs.close();
ps.close();
```

方法声明 2

```
InputStream getAsciiStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：以一字节 ASCII 字符流的形式返回传递数据库列值的 java 输入流；如果值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnLabel 无效，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

注意

在获取任何其他列的值之前必须读取返回流中的所有数据。下一次调用获取方法将隐式关闭该流。此外，当调用 InputStream.available 方法时，不管是否存在可用数据，流都可能返回 0。

示例

本示例调用了 ResultSet 对象的 getAsciiStrea 方法以 ASCII 字符流的形式获取字段值，并输出到控制台。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = rs.getAsciiStream("description");
    OutputStream os = System.out;
```

```
        int i;
        while(( i = is.read()) != -1){
            os.write(i);
        }
        is.close();
        os.flush();
        os.close();
    }
    rs.close();
    ps.close();
```

5.5.12 getBigDecimal 方法——获取 BigDecimal 型的数据

以具有全精度的 java.math.BigDecimal 的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
BigDecimal getBigDecimal(int columnIndex) throws SQLException
```

入口参数：columnIndex，第一个列是 1，第二个列是 2，…

返回值：列值（全精度）；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBigDecimal 方法以 BigDecimal 形式获取字段值，并输出到控制台。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    BigDecimal b = rs.getBigDecimal(2);
    System.out.println(b);
}
rs.close();
ps.close();
```

方法声明 2

```
BigDecimal getBigDecimal(String columnLabel) throws SQLException
```

入口参数：columnLabel 使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：列值（全精度）；如果值为 SQL NULL，则返回值为 java 编程语言

PDF电子书基地 http://dayo1982.400gb.com

中的 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBigDecimal 方法以 BigDecimal 形式获取字段值，并输出到控制台。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    BigDecimal b = rs.getBigDecimal("age");
    System.out.println(b);
}
rs.close();
ps.close();
```

5.5.13 getBinaryStream 方法——获取字节流型的数据

以未解释字节的流的形式获取此 ResultSet 对象的当前行中指定列的值。然后，可以按块从流中读取值。此方法尤其适合于获取很大的 LONGVARBINARY 值。

方法声明 1

```
InputStream getBinaryStream(int columnIndex) throws SQLException
```

入口参数：columnIndex 第一个列是 1，第二个列是 2，…

返回值：以未解释字节的流的形式返回传递数据库列值的 Java 输入流；如果值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBinaryStream 方法以未解释字节的流的形式获取字段值，并写到磁盘上。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    OutputStream os = new FileOutputStream(new File("c:\\photo.jpg"));
```

```
InputStream is = rs.getBinaryStream(4);                    //获取前行中的值
if(is != null){
    byte[] bs = new byte[is.available()];
    is.read(bs);
    os.write(bs);
    os.flush();
    is.close();
}
os.close();
}
rs.close();
ps.close();
```

方法声明 2

```
InputStream getBinaryStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：以未解释字节流的形式返回传递数据库列值的 Java 输入流；如果值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBinaryStream 方法以未解释字节的流的形式获取字段值，并写到磁盘上。

```
Connection conn = …                                        //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    OutputStream os = new FileOutputStream(new File("c:\\photo.jpg"));
    InputStream is = rs.getBinaryStream("photo");          //获取前行中的值
    if(is != null){
        byte[] bs = new byte[is.available()];
        is.read(bs);
        os.write(bs);
        os.flush();
        is.close();
    }
    os.close();
}
rs.close();
ps.close();
```


PDF电子书基地 http://dayo1982.400gb.com

5.5.14 getBlob 方法——获取 Blob 型的数据

以 Java 编程语言中 Blob 对象的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
Blob getBlob(int columnIndex) throws SQLException
```

入口参数：columnIndex，第一个列是 1，第二个列是 2，…

返回值：表示指定列中的 SQL BLOB 值的 Blob 对象。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBlob 方法以 Blob 对象的形式获取字段值。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Blob blob = rs.getBlob(4);
}
rs.close();
ps.close();
```

方法声明 2

```
Blob getBlob(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：表示指定列中 SQL BLOB 值的 Blob 对象。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getBlob 方法以 Blob 对象的形式获取字段值。

```
Connection conn = ……                                  //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Blob blob = rs.getBlob("photo");
```

```
    }
    rs.close();
    ps.close();
```

5.5.15 getBoolean 方法——获取布尔型的数据

该方法从 ResultSet 对象中获取 boolean 类型的数据。

方法声明 1

```
boolean getBoolean(int columnIndex)
                        throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getBoolean 方法，打印当指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT username,isPass FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getBoolean(2));
}
```

方法声明 2

```
boolean getBoolean(String columnLabel)
                        throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getBoolean 方法，打印当指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT username,isPass FROM table";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getBoolean("isPass"));
}
```

5.5.16 getByte 方法——获取字节型的数据

该方法从 ResultSet 对象中获取 byte 类型的数据。

PDF电子书基地 http://dayo1982.400gb.com

方法声明 1

```
byte getByte(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getByte 方法，打印当指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myBytes FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getByte(1));
}
```

方法声明 2

```
byte getByte(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getByte 方法，打印当指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myByte FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getByte("myByte"));
}
```

5.5.17 getBytes 方法——获取字节数组的数据

该方法从 ResultSet 对象中获取 byte 数组类型的数据。

方法声明 1

```
byte[] getBytes(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getByte 方法，得到当指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myBytes FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
if(rs.next()){
        Byte[] b = rs.getBytes(1);
}
```

方法声明 2

```
byte[] getBytes(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getByte 方法，得到当指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myBytes FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
if(rs.next()){
        byte[] b = rs.getBytes("myBytes");
}
```

5.5.18 getCharacterStream 方法——获取 Reader 型的数据

以 java.io.Reader 对象的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
Reader getCharacterStream(int columnIndex) throws SQLException
```

入口参数：columnIndex，第一个列是 1，第二个列是 2，…

返回值：包含列值的 java.io.Reader 对象；如果值为 SQL NULL，则返回值为 java 编程语言中的 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例调用了 ResultSet 对象的 getCharacterStream 方法以获取字段值赋给 Reader 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = rs.getCharacterStream(5);
}
```

方法声明 2

```
Reader getCharacterStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：包含列值的 java.io.Reader 对象；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getCharacterStream 方法以获取字段值赋给 Reader 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = rs.getCharacterStream("discription");
}
```

5.5.19 getClob 方法——获取 Clob 型的数据

以 java 编程语言中 Clob 对象的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
Clob getClob(int columnIndex) throws SQLException
```

入口参数：columnIndex 第一个列是 1，第二个列是 2，…

返回值：表示指定列中的 SQL Clob 值的 Clob 对象。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 getClob 方法以 Clob 对象的形式获取字段值。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Clob clob = rs.getClob(5);
}
```

方法声明 2

```
Reader getCharacterStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：表示指定列中的 SQL Clob 值的 Clob 对象。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 getClob 方法以 Clob 对象的形式获取字段值。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Clob clob = rs.getClob("discription");
}
```

5.5.20 getConcurrency 方法——获取并发模式

获取此 ResultSet 对象的并发模式。使用的并发由创建结果集的 Statement 对象确定。

方法声明

```
int getConcurrency() throws SQLException
```

返回值：并发类型，值为 ResultSet.CONCUR_READ_ONLY 或 ResultSet.

PDF电子书基地 http://dayo1982.400gb.com

CONCUR_UPDATABLE。

抛出异常：SQLException，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getConcurrenc 方法输出了并发类型的值。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
System.out.println(rs.getConcurrency()); //输出并发类型的值
```

5.5.21 getCursorName 方法——获取 SQL 光标的名称

获取此 ResultSet 对象使用的 SQL 光标的名称。在 SQL 中，通过指定的光标获取结果表。通过一个引用光标名称来定位的更新/删除语句，可以更新或删除结果集的当前行。为了确保光标具有支持更新的适当隔离级别，光标的 SELECT 语句的形式应该为 SELECT FOR UPDATE。如果省略 FOR UPDATE，则定位更新可能失败。

方法声明

```
String getCursorName() throws SQLException
```

返回值：此 ResultSet 对象的光标的 SQL 名称。

抛出异常

SQLException：如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 getCursorName 方法输出了 SQL 光标的名称。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    System.out.println(rs.getCursorName());
}
rs.close();
ps.close();
```

5.5.22 getDate 方法——获取日期型的数据

该方法从 ResultSet 对象中获取 java.sql.Date 类型的数据。

方法声明 1

```
Date getDate(int columnIndex)
                throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getDate 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myDate FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getDate(1));
}
```

方法声明 2

```
Date getDate(String columnLabel)
                throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getDate 方法打印指定列的值。

```
Connection conn = ……                                   //省略部分代码
String sql = "SELECT myDate FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getDate("myDate"));
}
```

方法声明 3

以 Java 编程语言中 java.sql.Date 对象的形式获取此 ResultSet 对象的当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Date getDate(int columnIndex, Calendar cal) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：在构造时间戳时使用的 java.util.Calendar 对象。

返回值：java.sql.Date 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getDat 方法使用指定的 Calendar 获取了数据库中的 Date 值并输出。

```
Connection conn = …                                          //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    cal = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("日期为： "+rs.getDate(1));
    Date date = rs.getDate(1, cal);
    System.out.println("美国日期为" + date);
}
rs.close();
ps.close();
```

方法声明 4

以 Java 编程语言中 java.sql.Date 对象的形式获取此 ResultSet 对象的当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Date getDate(String columnLabel, Calendar cal) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：java.sql.Date 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getDat 方法使用指定的 Calendar 获取了数据库中的 Date 值并输出。

```
Connection conn = …                                          //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    cal = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("日期为： "+rs.getDate(1));
    Date date = rs.getDate("birth", cal);
    System.out.println("美国日期为" + date);
}
rs.close();
ps.close();
```

5.5.23 getDouble 方法——获取双精度型的数据

该方法从 ResultSet 对象中获取 double 类型的数据。

方法声明 1

```
double getDouble(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getDouble 方法打印指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getDouble(1));
}
```

方法声明 2

```
double getDouble(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

PDF电子书基地 http://dayo1982.400gb.com

示例

下面的代码利用 getDouble 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getDouble("myNumber"));
}
```

5.5.24 getFetchDirection 方法——获取方向

获取此 ResultSet 对象的获取方向。

方法声明

```
int getFetchDirection() throws SQLException
```

返回值：此 ResultSet 对象的当前获取方向。

抛出异常：SQLException 如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getFetchDirection 方法输出了 ResultSet 对象获取方向的值。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
System.out.println(rs.getFetchDirection());
rs.close();
ps.close();
```

5.5.25 getFetchSize 方法——获取 ResultSet 对象的获取大小

获取此 ResultSet 对象的获取大小。

方法声明

```
int getFetchSize() throws SQLException
```

返回值：此 ResultSet 对象的当前获取大小。

抛出异常：SQLException，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getFetchSize 方法输出了 ResultSet 对象获取大小的值。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
System.out.println(rs.getFetchSize());
rs.close();
ps.close();
```

5.5.26 getFloat 方法——获取单精度型的数据

该方法从 ResultSet 对象中获取 float 类型的数据。

方法声明 1

```
float getFloat(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getFloat 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getFloat(1));
}
```

方法声明 2

```
float getFloat(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getFloat 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getFloat("myNumber"));
}
```


PDF电子书基地 http://dayo1982.400gb.com

5.5.27 getHoldability 方法——获取可保存性

获取此 ResultSet 对象的可保存性。

方法声明

```
int getHoldability() throws SQLException
```

返回值：

ResultSet.HOLD_CURSORS_OVER_COMMIT 或

ResultSet.CLOSE_CURSORS_AT_COMMIT。

抛出异常：SQLException，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getHoldability 方法输出了 ResultSet 对象可保存性的值。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    System.out.println(rs.getHoldability());
}
rs.close();
ps.close();
```

5.5.28 getInt 方法——获取整型数据

该方法从 ResultSet 对象中获取 int 类型的数据。

方法声明 1

```
int getInt(int columnIndex)
                throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getInt 方法打印指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getInt(1));
}
```

方法声明2

```
int getInt(String columnLabel)
                throws SQLException
```

入口参数：columnLabel是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用getInt方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getInt("myNumber"));
}
```

5.5.29 getLong方法——获取长整型数据

该方法从ResultSet对象中获取类long型的数据。

方法声明1

```
long getLong(int columnIndex)
                throws SQLException
```

入口参数：columnIndex是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用getLong方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getLong(1));
}
```

方法声明2

```
long getLong(String columnLabel)
                throws SQLException
```

入口参数：columnLabel是列名称或别名。

返回值：列值。

PDF电子书基地 http://dayo1982.400gb.com

抛出异常：SQLException。

示例

下面的代码利用 getLong 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getLong("myNumber"));
}
```

5.5.30 getMetaData 方法——获取对象的列信息

获取此 ResultSet 对象的列的编号、类型和属性。

方法声明

```
ResultSetMetaData getMetaData() throws SQLException
```

返回值：此 ResultSet 对象的列的描述。

抛出异常：SQLException 如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getMetaData 方法输出了 ResultSet 对象列的描述信息。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
ResultSetMetaData rsm = rs.getMetaData();              //获取属性信息
```

5.5.31 getNCharacterStream 方法——获取 Reader 形式的数据

以 java.io.Reader 对象的形式获取此 ResultSet 对象的当前行中指定列的值。访问 NCHAR、NVARCHAR 和 LONGNVARCHAR 列时可使用此方法。

方法声明 1

```
Reader getNCharacterStream(int columnIndex) throws SQLException
```

入口参数：columnIndex 第一个列是 1，第二个列是 2，…

返回值：包含该列值的 java.io.Reader 对象；如果该值为 SQL NULL，则返回 java 编程语言中的值 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关

闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码需要为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 getNCharacterStream 方法读取字段值，返回 Reader 对象，并写入到硬盘上。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
Reader r = null;
Writer w = new FileWriter("c:\\desc.txt");
if(rs.next()){
    r = rs.getNCharacterStream(6);
    int i;
    while((i = r.read()) != -1){
        w.write(i);
    }
    r.close();
}
w.close();
ps.close();
```

方法声明 2

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 为使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：包含该列值的 java.io.Reader 对象；如果该值为 SQL NULL，则返回 Java 编程语言中的值 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛出异常。

示例

本示例调用了 ResultSet 对象的 getClob 方法以 Clob 对象的形式获取字段值。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
Reader r = null;
Writer w = new FileWriter("c:\\desc.txt");
if(rs.next()){
      r = rs.getNCharacterStream("description");
      int i;
      while((i = r.read()) != -1){
            w.write(i);
      }
      r.close();
}
w.close();
ps.close();
```

5.5.32 getNClob 方法——获取 NClob 型的数据

以 Java 编程语言中 NClob 对象的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
NClob getNClob(int columnIndex) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

返回值：表示指定列中 SQL NCLOB 值的 NClob 对象。

抛出异常

SQLException：如果 columnIndex 无效，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 getNClob 方法读取字段值，返回 NClob 对象，NClob 获取输入流然后将数据写入到硬盘上。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
Writer w = new FileWriter("c:\\desc.txt");
if(rs.next()){
```

```
        NClob nclob = rs.getNClob(6);
        Reader r = nclob.getCharacterStream();
        int i;
        while((i = r.read()) != -1){
            w.write(i);
        }
        r.close();
    }
    w.close();
    ps.close();
```

方法声明 2

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：表示指定列中 SQL NCLOB 值的 NClob 对象。

抛出异常

SQLException：如果 columnLabel 无效，如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 getNClob 方法读取字段值，返回 NClob 对象，NClob 获取输入流，然后将数据写入硬盘。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
Writer w = new FileWriter("c:\\desc.txt");
if(rs.next()){
    NClob nclob = rs.getNClob("description");
    Reader r = nclob.getCharacterStream();
    int i;
    while((i = r.read()) != -1){
        w.write(i);
    }
    r.close();
}
w.close();
ps.close();
```


PDF电子书基地 http://dayo1982.400gb.com



5.5.33 getNString 方法——获取 String 型的数据

以 Java 编程语言中 String 的形式获取此 ResultSet 对象的当前行中指定列的值。访问 NCHAR、NVARCHAR 和 LONGNVARCHAR 列时可使用此方法。

方法声明 1

```
String getNString(int columnIndex) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

返回值：列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 getNString 方法读取字段值并输出。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    String name = rs.getNString(2);
    System.out.println(name);
}
ps.close();
```

方法声明 2

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：列值；如果该值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 getNString 方法读取字段值并输出。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    String name = rs.getNString("uname");
    System.out.println(name);
}
ps.close();
```

5.5.34 getObject 方法——获取 object 型的数据

该方法从 ResultSet 对象中获取 object 类型的数据。

方法声明 1

```
Object getObject(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getObject 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT username,myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getObject(1));
    System.out.println(rs.getObject(2));
}
```

方法声明 2

```
Object getObject(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

PDF电子书基地 http://dayo1982.400gb.com





示例

下面的代码利用 getObject 方法打印指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT username,myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
        System.out.println(rs.getObject("username"));
        System.out.println(rs.getObject("myNumber"));
}
```

5.5.35 getRow 方法——获取当前编号

获取当前行编号。第一行为 1 号，第二行为 2 号，依次类推。



方法声明

```
int getRow() throws SQLException
```

返回值：列值；如果该值为 SQL NULL，则返回值为 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。



注意

对于带有 TYPE_FORWARD_ONLY 的结果集类型的 ResultSet，对 getRow 方法的支持是可选的。



示例

本示例调用了 ResultSet 对象的 getRow 方法输出当前行编号。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users where id = 1";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
rs.absolute(5);
System.out.println(rs.getRow());
```

5.5.36 getShort 方法——获取短整型的数据

该方法从 ResultSet 对象中获取 short 类型的数据。



方法声明 1

```
short getShort(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getShort 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getShort(1));
}
```

方法声明 2

```
short getShort(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getShort 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myNumber FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getShort("myNumber"));
}
```

5.5.37 getStatement 方法——获取 Statement 对象

获取生成此 ResultSet 对象的 Statement 对象。如果结果集是以其他方式生成的（如通过 DatabaseMetaData 方法），则此方法可能返回 null。

方法声明

```
Statement getStatement() throws SQLException
```

返回值：生成此 ResultSet 对象的 Statment 对象；如果结果集是以其他方法生成的，则返回 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getStatement 方法获取了生成它的 Statement

PDF电子书基地 http://dayo1982.400gb.com

对象。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
Statement stmt = rs.getStatement();
```

5.5.38 getString 方法——获取 String 类型的数据

该方法从 ResultSet 对象中获取 String 类型的数据。

方法声明 1

```
String getString(int columnIndex)
                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getString 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT username,pwd FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getString(1));
    System.out.println(rs.getString(2));
}
```

方法声明 2

```
String getString(String columnLabel)
                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面代码利用 getString 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT username,pwd FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getString("username"));
    System.out.println(rs.getString("pwd"));
}
```

5.5.39 getTime 方法——获取时间型的数据

该方法从 ResultSet 对象中获取类 java.sql.Time 型的数据。

方法声明 1

```
Time getTime(int columnIndex)
                throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getTime 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myTime FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getTime(1));
}
```

方法声明 2

```
Time getTime(String columnLabel)
                throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getTime 方法打印指定列的值。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT myTime FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
      System.out.println(rs.getTime("myTime"));
}
```

方法声明 3

以 Java 编程语言中 java.sql.Time 对象的形式获取此 ResultSet 对象的当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Time getTime(int columnIndex, Calendar cal) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：在构造时间戳时使用的 java.util.Calendar 对象。

返回值：java.sql.Time 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getTime 方法使用指定的 Calendar 获取了数据库中的 Time 值并输出。

```
Connection conn = …                                          //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    cal = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("时间为： "+rs.getTime(1));
    Time time = rs.getTime(1, cal);
    System.out.println("美国时间为" + time);
}
rs.close();
ps.close();
```

方法声明 4

以 Java 编程语言中 java.sql.Time 对象的形式获取此 ResultSet 对象的当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Time getTime (int columnIndex, Calendar cal) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：java.sql.Time 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getTime 方法使用指定的 Calendar 获取了数据库中的 Time 值并输出。

```
Connection conn = …                                          //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    cal = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("时间为： "+rs.getTime(1));
    Time time = rs.getTime("birth", cal);
    System.out.println("美国时间为" + time);
}
rs.close();
ps.close();
```

5.5.40 getTimestamp 方法——获取时间戳

该方法从 ResultSet 对象中获取 java.sql.Timestap 类型的数据。

方法声明 1

```
Timestamp getTimestamp(int columnIndex)
                                    throws SQLException
```

入口参数：columnIndex 是列序号。

返回值：列值。

抛出异常：SQLException。

示例

下面的代码利用 getTimestamp 方法打印指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myDateTime FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getTimestamp(1));
}
```

方法声明 2

```
Timestamp getTimestamp(String columnLabel)
                                    throws SQLException
```

入口参数：columnLabel 是列名称或别名。

返回值：列值。

抛出异常：SQLException。

PDF电子书基地 http://dayo1982.400gb.com

示例

下面的代码利用 getTimestamp 方法打印指定列的值。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT myDateTime FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while(rs.next()){
    System.out.println(rs.getTimestamp("myDateTime "));
}
```

方法声明 3

以 Java 编程语言中 java.sql.Timestamp 对象的形式获取此 ResultSet 对象的当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Timestamp getTimestamp(int columnIndex, Calendar cal) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：在构造时间戳时使用的 java.util.Calendar 对象。

返回值：java.sql.Timestamp 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getTimestamp 方法使用指定的 Calendar 获取了数据库中的 Timestamp 值并输出。

```
Connection conn = …                                          //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    c = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("日期为：  "+rs.getTimestamp(1));
    Timestamp date = rs.getTimestamp(1, c);
    System.out.println("美国日期为"+date);
}
rs.close();
ps.close();
```

方法声明 4

以 Java 编程语言中 java.sql.Timestamp 对象的形式获取此 ResultSet 对象的

当前行中指定列的值。如果底层数据库未存储时区信息，则此方法使用给定日历构造时间戳的适当毫秒值。

```
Timestamp getTimestamp(int columnIndex, Calendar cal) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

cal：使用 SQL AS 子句指定的列标签；如果未指定 SQL AS 子句，则标签是列名称。

返回值：java.sql.Timestamp 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

示例

本示例调用了 ResultSet 对象的 getTimestamp 方法使用指定的 Calendar 获取了数据库中的 Timestamp 值并输出。

```
Connection conn = ……                                    //省略部分代码
String sql = "select birth from users where id = 2";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    //构造一个美国时区的 Calendar 对象
    Calendar    c = Calendar.getInstance(TimeZone.getTimeZone("America"));
    System.out.println("日期为： "+rs.getTimestamp(1));
    Timestamp date = rs.getTimestamp("birth", c);
    System.out.println("美国日期为"+date);
}
rs.close();
ps.close();
```

5.5.41 getType 方法——获取 ResultSet 对象的类型

该方法用于获取 ResultSet 对象的类型，类型由创建结果集的 Statement 对象确定。

方法声明

```
int getType()
    throws SQLException
```

返回值：ResultSet 接口定义的常量。

抛出异常：SQLException。

示例

下面的代码利用 getType 方法获取 ResultSet 对象的类型。

PDF电子书基地 http://dayo1982.400gb.com

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM myTable";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
int i = rs.getType();                                  //得到 ResultSet 对象的类型
```

5.5.42 getURL 方法——获取 URL 对象形式的数据

以 Java 编程语言中 java.net.URL 对象的形式获取此 ResultSet 对象的当前行中指定列的值。

方法声明 1

```
URL getURL(int columnIndex) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

返回值：java.net.URL 对象形式的列值；如果值为 SQL NULL，则返回值为 java 编程语言中的 null。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 getURL 方法获取了 java.net.URL 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    URL url = rs.getURL(9);
    System.out.println(url.getHost());
}
```

方法声明 2

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

入口参数：columnLabel 使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

返回值：java.net.URL 对象形式的列值；如果值为 SQL NULL，则返回值为 Java 编程语言中的 null。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误或在已关

闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 getURL 方法获取了 java.net.URL 对象。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    URL url = rs.getURL("blog");
    System.out.println(url.getHost());
}
```

5.5.43 getWarnings 方法——获取报告的第一个警告

获取此 ResultSet 对象上的调用报告的第一个警告。此 ResultSet 对象上的后续警告会被链接到此方法返回的 SQLWarning 对象。

方法声明

```
SQLWarning getWarnings() throws SQLException
```

返回值：报告的第一个 SQLWarning 对象；如果不存在，则返回 null。

抛出异常：SQLException，如果 columnIndex 无效；如果发生数据库访问错误或在已关闭的结果集上调用此方法。

注意

每次读取新行时，都会自动清除警告链。此警告链仅包含 ResultSet 方法产生的警告。Statement 方法（如读取 OUT 参数）产生的任何警告都将链接在 Statement 对象上。

示例

本示例调用了 ResultSet 对象的 getWarnings 方法获取了 SQLWarning。

```
Connection conn = …                                    //省略部分代码
SString sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
SQLWarning waring = rs.getWarnings();
```

5.5.44 insertRow 方法——插入当前行到数据库中

将插入行的内容插入此 ResultSet 对象和数据库中。调用此方法时，光标必须位于插入行上。

方法声明

```
void insertRow() throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

抛出异常

如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法；如果在光标不位于插入行上时调用此方法，或者插入行中所有不可为 null 的列中还存在未分配非 null 值的列。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 ResultSet 对象中的 insertRow 方法在数据库中插入一条数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
rs.moveToInsertRow();                                        //将光标移动到插入行
rs.updateInt("id", 3);
rs.updateString("uname", "明日科技");
rs.insertRow();                                              //插入数据
rs.close();
ps.close();
```

5.5.45 isAfterLast 方法——判断光标是否位于最后一行之后

该方法用于判断光标是否位于 Request 对象最后一行之后。

方法声明

```
boolean isAfterLast()
        throws SQLException
```

返回值：光标如果位于最后一行之后，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码判断光标是否位于最后一行之后。

```
Connection conn = …                                          //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
System.out.println(rs.isAfterLast());                        //光标没在最后一行之后打印 false
rs.afterLast();                                              //将光标移动到最后一行之后
System.out.println(rs.isAfterLast());                        //打印 true
```

5.5.46 isBeforeFirst 方法——判断光标是否位于第一行之前

该方法用于判断光标是否位于 ResultSet 对象的第一行之前。

方法声明

```
boolean isBeforeFirst()
        throws SQLException
```

返回值：光标位于 ResultSet 对象的第一行之前，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面代码通过 isBeforeFirst 方法判断光标是否在 ResultSet 对象第一行之前。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
System.out.println(rs.isBeforeFirst());                //光标没在第一行之前打印 true
rs.next();                                             //移动光标
System.out.println(rs.isAfterLast());                  //打印 false
```

5.5.47 isClosed 方法——判断 ResultSet 对象是否已经关闭

该方法用于判断 ResultSet 对象是否已经关闭。

方法声明

```
boolean isClosed()
        throws SQLException
```

返回值：如果 ResultSet 对象已经关闭，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 isClosed 方法判断 ResultSet 对象是否已经被关闭。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
System.out.println(rs.isClosed());                     //在没有关闭之前打印 false
rs.close();                                            //将 ResultSet 对象关闭
System.out.println(rs.isClosed());                     //关闭之后打印 true
```

5.5.48 isFirst 方法——判断光标是否位于第一行

该方法用于判断光标是否位于 ResultSet 对象的第一行。

方法声明

```
boolean isFirst()
        throws SQLException
```

返回值：光标位于 ResultSet 对象的第一行，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 isFirst 方法判断光标是否在第一行。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
```


PDF电子书基地 http://dayo1982.400gb.com

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
System.out.println(rs.isFirst());                    //光标不在第一行打印 false
rs.first();                                          //将光标移动到第一行
System.out.println(rs.isFirst());                    //打印 true
```

5.5.49 isLast 方法——判断光标是否位于最后一行

该方法用于判断光标是否位于此 ResultSet 对象的最后一行。

方法声明

```
boolean isLast()
        throws SQLException
```

返回值：如果光标位于最后一行上，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 isLast 方法判断光标是否在 ResultSet 对象的最后一行。

```
Connection conn = …                                  //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
System.out.println(rs.isLast ());                    //光标不在最后一行打印 false
rs.last ();                                          //将光标移动到最后一行
System.out.println(rs.isLast());                     //打印 true
```

5.5.50 last 方法——将光标移动到最后一行

该方法用于将光标移动到此 ResultSet 对象的最后一行。

方法声明

```
boolean last()
        throws SQLException
```

返回值：如果光标在有效的行上，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 last 方法将光标移到 ResultSet 对象的最后一行。

```
Connection conn = …                                  //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
rs.last ();                                          //将光标移动到最后一行
```

5.5.51 moveToCurrentRow 方法——将光标移动到记住的光标位置

将光标移动到记住的光标位置，通常为当前行。如果光标不位于插入行上，则此方法无效。

方法声明

```
void moveToCurrentRow() throws SQLException
```

抛出异常

如果发生数据库访问错误；在已关闭的结果集上调用此方法或结果集并发性为 CONCUR_READ_ONLY。

SQLFeatureNotSupportedException： 如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 ResultSet 对象中的 moveToCurrentRow 方法将光标移动到记住的光标位置。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
rs.next();
rs.moveToCurrentRow();
```

5.5.52 moveToInsertRow 方法——将光标移动到插入行

将光标移动到插入行。将光标置于插入行上时，当前的光标位置会被记住。插入行是一个与可更新结果集相关联的特殊行。它实际上是一个缓冲区，在将行插入到结果集前可以通过调用更新方法在其中构造新行。当光标位于插入行上时，仅能调用更新方法、获取方法以及 insertRow 方法。每次在调用 insertRow 之前调用此方法时，必须为结果集中的所有列分配值。在对列值调用获取方法之前，必须调用更新方法。

方法声明

```
void moveToInsertRow() throws SQLException
```

抛出异常

如果发生数据库访问错误；在已关闭的结果集上调用此方法或结果集并发性为 CONCUR_READ_ONLY。

SQLFeatureNotSupportedException： 如果 JDBC 驱动程序不支持此方法。

示例

本示例调用 ResultSet 对象中的 moveToInsertRow 方法将光标移动动到插入行然后插入一条数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
```


PDF电子书基地 http://dayo1982.400gb.com

```
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
rs.moveToInsertRow();                                    //将光标移动到插入行
rs.updateInt("id", 3);
rs.updateString("uname", "明日科技");
rs.insertRow();                                          //插入数据
rs.close();
ps.close();
```

5.5.53 next 方法——将光标从当前的位置向下移动一行

该方法用于将光标从当前的位置向下移动一行，当调用 next 方法返回 false 时说明当前光标在最后一行的后面。

方法声明

```
boolean next()
                throws SQLException
```

返回值：如果新的当前行有效，则返回 true，否则返回 false。

抛出异常：SQLException。

示例

下面的代码利用 next 方法打印 ResultSet 对象中的数据。

```
try{
    Class.forName("com.mysql.jdbc.Driver");                  //数据库驱动
    String url = "jdbc:mysql://localhost:3306/test";
    String name = "root";
    String pwd = "root";
    String sql = "SELECT username FROM users";
    conn=DriverManager.getConnection(url,name,pwd);
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery(sql);
    while(rs.next()){                                        //将光标向下移动一行
        System.out.println(rs.getString("username"));
    }
    rs.close();
    st.close();
}catch(Exception e){
    e.printStackTrace();
}finally{
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        conn = null;
    }
}
```

5.5.54 previous 方法——光标移动到上一行

该方法用于将光标移动到此 ResultSet 对象的上一行。

方法声明

```
boolean previous()
        throws SQLException
```

返回值：如果光标位于有效行上，则返回 true，否则光标位于第一行的上一行返回 false。

抛出异常：SQLException。

示例

下面的代码利用 previous 方法将光标向上移动。

```
Connection conn = …                                    //省略部分代码
String sql = "SELECT * FROM users";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
//将光标向上移动一行，由于光标已经在第一行之前，所以打印 false
System.out.println(rs.previous());
```

5.5.55 relative 方法——按相对行数移动光标

按相对行数（或正或负）移动光标。试图移动到结果集的第一行/最后一行之外，会将光标置于第一行之前或最后一行之后。调用 relative(0)有效，但是不更改光标位置。

方法声明

```
boolean relative(int rows) throws SQLException
```

入口参数：row 指定从当前行开始移动的行数的 int；正数表示光标正向移动；负数表示光标反向移动。

返回值：如果光标移动到此 ResultSet 对象的位置处，则返回 true；如果光标在第一行的前面或最后一行的后面，则返回 false。

抛出异常：SQLException，如果发生数据库访问错误；在已关闭的结果集上调用此方法或结果集类型为 TYPE_FORWARD_ONLY。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例利用 absolute 方法将光标移到 Request 对象第 5 行，然后向后移动 5 行，又向前移动 3 行。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
```


PDF电子书基地 http://dayo1982.400gb.com

```
ResultSet rs = ps.executeQuery();
rs.absolute(5);                          //光标移动到第 5 行
System.out.println(rs.getRow());
rs.relative(5);                          //光标向后移动 5 行
System.out.println(rs.getRow());
rs.relative(-3);                         //光标移向前动 3 行
System.out.println(rs.getRow());
```

5.5.56 setFetchDirection 方法——设置行的处理方向

设置此 ResultSet 对象中行的处理方向。初始值由生成此 ResultSet 对象的 Statement 对象确定。获取方向可以在任何时间更改。

方法声明

```
void setFetchDirection(int direction) throws SQLException
```

入口参数：direction 指定建议获取方向的 int；ResultSet.FETCH_FORWARD、ResultSet.FETCH_REVERSE 或 ResultSet.FETCH_UNKNOWN 之一。

抛出异常：SQLException，如果发生数据库访问错误，在已关闭的结果集上调用此方法或结果集类型 TYPE_FORWARD_ONLY，但获取方向不是 FETCH_FORWARD。

示例

本示例利用 setFetchDirection 方法设置行的处理方向。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
rs.setFetchDirection(ResultSet.FETCH_FORWARD);
```

5.5.57 setFetchSize 方法——设置结果集的最大行数

为 JDBC 驱动程序设置此 ResultSet 对象需要更多行时应该从数据库获取的行数。如果指定的获取大小为零，则 JDBC 驱动程序忽略该值，随意对获取大小作出它自己的最佳猜测。默认值由创建结果集的 Statement 对象设置。获取大小可以在任何时间更改。

方法声明

```
void setFetchSize(int rows) throws SQLException
```

入口参数：rows 要获取的行数。

抛出异常：SQLException，如果发生数据库访问错误；在已关闭的结果集上调用此方法或不满足条件 rows ≥ 0。

示例

本示例利用 setFetchSize 方法设置了需要更多行时应该从数据库获取的行数。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
rs.setFetchSize(10);
```

5.5.58 updateAsciiStream 方法——用 ascii 流值更新指定的列

用 ascii 流值更新指定的列。将根据需要从流中读取数据，一直读取到流末尾。更新方法用于更新当前行或插入行中的列值。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateAsciiStream(int columnIndex, InputStream x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateAsciiStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\desc.txt");
    rs.updateAsciiStream(5, is);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 2

用 ascii 流值更新指定列，该列将具有指定字节数。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或

PDF电子书基地 http://dayo1982.400gb.com

insertRow 方法。

```
void updateAsciiStream(int columnIndex, InputStream x, int length)
    throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\desc.txt");
    int length = is.available();
    rs.updateAsciiStream(5, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 3

用 ascii 流值更新指定列，该列将具有指定字节数。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateAsciiStream(int columnIndex, InputStream x, long length)
    throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\desc.txt");
    long length = is.available();
    rs.updateAsciiStream(5, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 4

用 ascii 流值更新指定列。将根据需要从流中读取数据，一直读取到流末尾。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateAsciiStream(String columnLabel, InputStream x)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateAsciiStream 方法更新了光标所在行的

PDF电子书基地 http://dayo1982.400gb.com

数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\desc.txt");
    rs.updateAsciiStream("description", is);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 5

用 ascii 流值更新指定列，该列将具有指定字节数。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateAsciiStream(String columnIndex, InputStream x, int length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
```

```
        InputStream is = new FileInputStream("c:\\desc.txt");
        int length = is.available();
        rs.updateAsciiStream("description", is, length);
        rs.updateRow();
        System.out.println("更新成功!!! ");
    }
rs.close();
ps.close();
```

方法声明 6

用 ascii 流值更新指定列，该列将具有指定字节数。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateAsciiStream(String columnIndex, InputStream x, long length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\desc.txt");
    long length = is.available();
    rs.updateAsciiStream("description", is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```


PDF电子书基地 http://dayo1982.400gb.com

5.5.59 updateBigDecimal 方法——用 BigDecimal 值更新指定列

用 java.math.BigDecimal 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    rs.updateBigDecimal(3, new BigDecimal(100));
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateBigDecimal(String columnLabel, BigDecimal x)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集

并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    rs.updateBigDecimal("age", new BigDecimal(100));
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.60 updateBinaryStream 方法——用二进制流值更新指定的列

用二进制流值更新指定的列。将根据需要从流中读取数据，一直读取到流末尾。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateBinaryStream(int columnIndex, InputStream x)
    throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，……

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
```


PDF电子书基地 http://dayo1982.400gb.com

```
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    rs.updateBinaryStream(6, is);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 2

用二进制流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(int columnIndex, InputStream x, int length)
        throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    int length = is.available();
    rs.updateBinaryStream(6, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 3

用二进制流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用.updateRow 或 insertRow 方法。

```
void updateBinaryStream(int columnIndex, InputStream x, long length)
        throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，……

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    long length = is.available();
    rs.updateBinaryStream(6, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 4

用二进制流值更新指定的列。将根据需要从流中读取数据，一直读取到流末尾。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(String columnLabel, InputStream x)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，

PDF电子书基地 http://dayo1982.400gb.com

则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new EileInputStream("c:\\photo.jpg");
    rs.updateBinaryStream("photo", is);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 5

用二进制流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(String columnIndex, InputStream x, int length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    int length = is.available();
    rs.updateBinaryStream("photo", is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 6

用二进制流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(String columnIndex, InputStream x, long length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBinaryStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
```


PDF电子书基地 http://dayo1982.400gb.com

```
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    long length = is.available();
    rs.updateBinaryStream("photo", is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

5.5.61 updateBlob 方法——用 Blob 值更新指定列

用 java.sql.Blob 值更新指定列。更新方法用于更新当前行或插入行中的列值。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateBlob(int columnIndex, Blob x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    byte[] bs = new byte[is.available()];
    is.read(bs);
    Blob blob = new SerialBlob(bs);
    rs.updateBlob(6, blob);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 2

使用给定输入流值更新指定列。将根据需要从流中读取数据，一直读取到流末尾。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBlob(int columnIndex, InputStream x, int length)
        throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    int length = is.available();
    rs. updateBlob(6, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 3

使用给定输入流更新指定的列，该流将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBlob(int columnIndex, InputStream x, long length)
        throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

PDF电子书基地 http://dayo1982.400gb.com

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    long length = is.available();
    rs. updateBlob(6, is, length);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 4

用 java.sql.Blob 值更新指定列。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(String columnLabel, InputStream x)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
```

```
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    byte[] bs = new byte[is.available()];
    is.read(bs);
    Blob blob = new SerialBlob(bs);
    rs.updateBlob("photo", blob);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 5

使用给定输入流值更新指定列。将根据需要从流中读取数据，一直读取到流末尾。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBlob(String columnIndex, InputStream x, int length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    int length = is.available();
```


PDF电子书基地 http://dayo1982.400gb.com

```
        rs. updateBlob("photo", is, length);
        rs.updateRow();
        System.out.println("更新成功!!!  ");
    }
    rs.close();
    ps.close();
```

方法声明 6

使用给定输入流更新指定的列，该流将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBlob(String columnIndex, InputStream x, long length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBlob 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    InputStream is = new FileInputStream("c:\\photo.jpg");
    long length = is.available();
    rs. updateBlob("photo", is, length);
    rs.updateRow();
    System.out.println("更新成功!!!  ");
}
rs.close();
```

5.5.62 updateBoolean 方法——用 boolean 值更新指定列

用 boolean 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateBoolean(int columnIndex, boolean x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBoolean 方法更新了光标所在行的布尔型数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    rs.updateBoolean(4, false);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateBoolean(String columnLabel, boolean x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBigDecimal 方法更新了光标所在行的

PDF电子书基地 http://dayo1982.400gb.com

数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    rs.updateBoolean("marryed", false);
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.63 updateByte 方法——用 byte 值更新指定列

用 byte 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateByte(int columnIndex, byte x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateByte 方法更新了光标所在行的数据。

```
Connection conn = ……                                  //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    byte b = 30;
    rs.updateByte(3, b);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateBytes(String columnLabel, byte[] x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateByte 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    byte b = 30;
    rs.updateByte("age", b);
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.64 updateBytes 方法——用 byte 数组值更新指定列

用 byte 数组值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateBytes(int columnIndex, byte[] x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

PDF电子书基地 http://dayo1982.400gb.com

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBytes 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    InputStream is = new FileInputStream("c:\\123.jpg");
    byte[] bs = new byte[is.available()];
    is.read(bs);
    rs.updateBytes(5, bs);
    rs.updateRow();
    is.close();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateBytes(String columnLabel, byte[] x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateBytes 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.absolute(5)){
    InputStream is = new FileInputStream("c:\\123.jpg");
    byte[] bs = new byte[is.available()];
    is.read(bs);
```

```
        rs.updateBytes("photo", bs);
        rs.updateRow();
        is.close();
    }
    rs.close();
    ps.close();
```

5.5.65 updateCharacterStream 方法——用二进制流值更新指定的列

用二进制流值更新指定的列。将根据需要从流中读取数据，一直读取到流末尾。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateCharacterStream(int columnIndex, Reader x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    rs.updateCharacterStream(5, r);
    rs.updateRow();
    System.out.println("更新成功!!! ");
    }
    rs.close();
    ps.close();
```

方法声明 2

用字符流值更新指定列，该列将具有指定字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

PDF电子书基地 http://dayo1982.400gb.com

```
void updateCharacterStream(int columnIndex, Reader x, int length)
    throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    rs.updateCharacterStream(5, r, 20);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 3

用字符流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateCharacterStream(int columnLabel, Reader reader, long length)
    throws SQLException
```

入口参数

columnLabel：第一个列是 1，第二个列是 2，…

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集

并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    long l = 20;
    rs.updateCharacterStream(5, r, l);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 4

用字符流值更新指定的列。将根据需要从流中读取数据，一直读取到流末尾。更新方法用于更新当前行或插入行中的列值。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateBinaryStream(String columnLabel, InputStream x)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
```


PDF电子书基地 http://dayo1982.400gb.com

```
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    rs.updateCharacterStream("description", r);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明 5

用字符流值更新指定列，该列将具有指定字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateCharacterStream(String columnIndex, Reader x, int length)
    throws SQLException
```

入口参数

columnIndex：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    rs.updateCharacterStream("description", r, 20);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

方法声明6

用字符流值更新指定的列，该值将具有指定的字节数。更新方法不更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

```
void updateCharacterStream(String columnLabel, Reader reader, long length)
    throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

length：流的长度。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateCharacterStream 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Reader r = new FileReader("c:\\desc.txt");
    long l = 20;
    rs.updateCharacterStream("description", r, l);
    rs.updateRow();
    System.out.println("更新成功!!! ");
}
rs.close();
ps.close();
```

5.5.66 updateDate 方法——用 Date 值更新指定列

用 java.sql.Date 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明1

```
void updateDate(int columnIndex, Date x) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateDate 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
      Date date = new Date(new java.util.Date().getTime());
      rs.updateDate(6, date);
      rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateDate(String columnLabel, Date x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateDate 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
```

```
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Date date = new Date(new java.util.Date().getTime());
    rs.updateDate("birth", date);
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.67 updateDouble 方法——用 double 值更新指定列

用 double 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateDouble(int columnIndex, double x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateDouble 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateDouble(9, 123.456d);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateDouble(String columnLabel, double x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，

PDF电子书基地 http://dayo1982.400gb.com

则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateDouble 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
      rs.updateDouble("number", 123.456d);
      rs.updateRow();
}
rs.close();
ps.close();
```

5.5.68 updateFloat 方法——用 float 值更新指定列

用 float 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateFloat(int columnIndex, float x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateFloat 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
```

```
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs. updateFloat(9, 123.456d);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateFloat(String columnLabel, float x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateFloat 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs. updateFloat("number", 123.456f);
    rs.updateRow();
}
rs.close();
```

5.5.69 updateInt 方法——用 int 值更新指定列

用 int 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateInt(int columnIndex, int x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，……

PDF电子书基地 http://dayo1982.400gb.com

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateInt 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateInt(3, 50);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateInt(String columnLabel, int x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateInt 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateInt("age", 50);
```

```
        rs.updateRow();
    }
    rs.close();
    ps.close();
```

5.5.70 updateLong 方法——用 long 值更新指定列

用 long 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateLong(int columnIndex, long x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateLong 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateLong(3, 50);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateLong(String columnLabel, long x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集

PDF电子书基地 http://dayo1982.400gb.com

并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateLong 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateLong("age", 50);
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.71 updateNString 方法——用 String 值更新指定列

用 String 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateNString(int columnIndex, String nString) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 updateNString 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
```

```
if(rs.next()){
    rs.updateNString(8, "明日科技编程词典");
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateNString(String columnLabel, String nString) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

注意

数据库编码须为 UTF-8，否则抛异常。

示例

本示例调用了 ResultSet 对象的 updateLong 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateNString("description", "明日科技编程词典");
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.72 updateNull 方法——用 null 值更新指定列

用 null 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateNull(int columnIndex) throws SQLException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateNull 方法更新了光标所在行的数据为 Null 值。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateNull(8);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateNull(String columnLabel) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateNull 方法更新了光标所在行的数据为 Null 值。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
```

```
if(rs.next()){
    rs.updateNull("description");
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.73 updateRow 方法——更新当前行

用此 ResultSet 对象的当前行的新内容更新底层数据库。光标不位于插入行上时不能调用此方法。

方法声明

```
void updateRow() throws SQLException
```

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateRow 方法将当前行的新内容更新底层数据库。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateString("uname", "明日科技");
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.74 updateShort 方法——用 short 值更新指定列

用 short 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateShort(int columnIndex, short x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，……

x：新列值。

PDF电子书基地 http://dayo1982.400gb.com

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateShort 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    short age = 20;
    rs.updateShort(3, age);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateShort(String columnLabel, short x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateShort 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    short age = 20;
    rs.updateShort("age", age);
```

```
        rs.updateRow();
    }
    rs.close();
    ps.close();
```

5.5.75 updateString 方法——用 String 值更新指定列

用 String 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateString(int columnIndex, String x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，……

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateShort 方法更新了光标所在行的数据。

```
Connection conn = …                                    //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateString(2, "明日科技");
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateString(String columnLabel, String x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集

PDF电子书基地 http://dayo1982.400gb.com

并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateShort 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    rs.updateString("uname", "明日科技");
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.76 updateTime 方法——用 Time 值更新指定列

用 java.sql.Time 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateTime(int columnIndex, Time x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLException：如果 columnIndex 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateTime 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users where id = 149";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Time time = new Time(new Date().getTime());
    rs.updateTime(10, time);
```

```
            rs.updateRow();
        }
        rs.close();
        ps.close();
```

方法声明 2

```
void updateTime(String columnLabel, Time x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateTime 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users where id = 149";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Time time = new Time(new Date().getTime());
    rs.updateTime("regtime", time);
    rs.updateRow();
}
rs.close();
ps.close();
```

5.5.77 updateTimestamp 方法——用时间戳更新指定列

用 java.sql.Timestamp 值更新指定列。更新方法用于更新当前行或插入行中的列值，并不会更新底层数据库；更新数据库要调用 updateRow 或 insertRow 方法。

方法声明 1

```
void updateTimestamp(int columnIndex, Timestamp x) throws SQLException
```

入口参数

columnIndex：第一个列是 1，第二个列是 2，…

x：新列值。

抛出异常

SQLExceptionL：如果 columnIndex 无效；如果发生数据库访问错误；结果

PDF电子书基地 http://dayo1982.400gb.com

集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateTimestamp 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Timestamp time = new Timestamp(new Date().getTime());
    rs.updateTimestamp(7, time);
    rs.updateRow();
}
rs.close();
ps.close();
```

方法声明 2

```
void updateTimestamp(String columnLabel, Timestamp x) throws SQLException
```

入口参数

columnLabel：使用 SQL AS 子句指定的列标签。如果未指定 SQL AS 子句，则标签是列名称。

x：新列值。

抛出异常

SQLException：如果 columnLabel 无效；如果发生数据库访问错误；结果集并发性为 CONCUR_READ_ONLY 或在已关闭的结果集上调用此方法。

SQLFeatureNotSupportedException：如果 JDBC 驱动程序不支持此方法。

示例

本示例调用了 ResultSet 对象的 updateTimestamp 方法更新了光标所在行的数据。

```
Connection conn = …                                          //省略部分代码
String sql = "select * from users";
PreparedStatement ps = conn.prepareStatement(sql,
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet rs = ps.executeQuery();
if(rs.next()){
    Timestamp time = new Timestamp(new Date().getTime());
    rs.updateTimestamp("birth", time);
    rs.updateRow();
}
rs.close();
ps.close();
```

第6章　JSP指令

6.1　include指令——包含文件指令

该指令用于当前的JSP页面，在使用该指令的位置嵌入其他的文件，如果被包含的文件中有可执行的代码，则包含代码执行后的结果。

语法

```
<%@ include file="文件的绝对路径或相对路径" %>
```

参数说明：

file属性：该属性指定被包含的文件，使用include指令时，它必须存在。该属性不支持任何表达式，也不允许通过如下的方式来传递参数。

示例

```
<%@ include file="welcome.jsp?name=yxq" %>
```

注意

如果该属性值以“/”开头，那么指定的是一个绝对路径，将在当前应用的根目录下查找文件；如果是以文件名称或文件夹名开头，那么指定的是一个相对路径，将在当前页面的目录下查找文件。

6.2　Page指令——页面指令

Page指令即页面指令，利用页面指令可以定义在整个JSP页面范围内有效的属性，其使用格式如下：

```
<%@ page attribute1="value1" attribute2="value2"...%>
```

page指令可以放在JSP页面中的任意行，但习惯上放在文件的开始部分，便于程序代码的阅读。Page指令具有多种属性，通过这些属性的设置可以影响到当前的JSP页面。例如，在页面中正确设置当前页面响应的MIME类型为“text/html”，如果MIME类型设置不正确，则当服务器将数据传输给客户端进行显示时，客户端将无法识别传送来的数据，从而不能正确地显示内容。

Page指令中包含了许多的属性。除import属性外，其他属性只能在指令中出现一次。Page指令具有的属性如下。

语法

```
<%@ page
        [ language="java" ]
```


PDF电子书基地 http://dayo1982.400gb.com

```
[ contentType="mimeType;charset=CHARSET" ]
[ import="{package.class|pageage.*},..." ]
[ extends="package.class" ]
[ session="true|false" ]
[ buffer="none|8kb|size kb ]
[ autoFlush="true|false" ]
[ isThreadSafe="true|false" ]
[ info="text" ]
[ errorPage="relativeURL" ]
[ isErrorPage="true|false" ]
[ isELIgnored="true|false" ]
[ pageEncoding="CHARSET" ]
%>
```

面对 Page 指令具有如此多的属性，在实际编程时，程序员并不需要一一列出。其中很多的属性可以忽略掉，此时 Page 指令将使用这些属性的默认值来设置 JSP 页面。

6.2.1 autoFlush 属性——是否自动输出缓冲区的内容

该属性默认值为 true，表示当缓冲区已满时，自动将其中的内容输出到客户端。如果设为 false，则当缓冲区中的内容超出其设置的大小时，会产生“JSP Buffer overflow”溢出异常。

示例

```
<%@page autoFlush="true" %>
```

注意

若 buffer 属性设为 none，则 autoFlush 不能设为 false。

6.2.2 buffer 属性——设置的缓冲区大小

设置了 out 对象（JspWriter 类型）使用的缓冲区的大小。若设置为“none”，表示不使用缓存，而直接通过 PrintWriter 对象进行输出；如果将该属性指定为数值，则输出缓冲区的大小不应小于该值。默认值为 8KB（因不同的服务器而不同，但大多数情况下都为 8KB）。

示例

```
<%@page buffer="8kb" %>
```

6.2.3 language 属性——设置 JSP 脚本使用的语言

设置当前页面中编写 JSP 脚本使用的语言，默认值为“java”。

示例

```
<%@ page language="java" %>
```

上述代码设置了当前页面中使用 Java 语言来编写 JSP 脚本，目前只能是 Java。

6.2.4 contentType属性——设置页面响应的MIME类型

设置页面响应的MIME类型，通常被设置为“text/html”。

示例

```
<%@ page contentType="text/html" %>
```

如果类属性设置不正确，如设置为“text/css”，那么客户端浏览器在显示HTML样式时，不能对HTML标记进行解释，而显示HTML代码。

示例

在该属性中还可以设置JSP字符的编码。

```
<%@ page contentType="text/html;charset=gb2312" %>
```

默认的编码为“ISO-8859-1”。

6.2.5 errorPage属性——指定异常处理页面

该属性用来指定一个当前页面出现异常时所要调用的页面。如果属性值是以“/”开头的路径，则将在当前应用程序的根目录下查找文件；否则，将在当前页面的目录下查找文件。

示例

```
<%@page errorPage="error.jsp" %>
```

6.2.6 extends属性——指定页面转换为Servlet后继承的类

用于指定一个JSP页面转换为Servlet后继承的类。在JSP中通常不会设置该属性，JSP容器会提供转换后的Servlet继承的父类。并且如果设置了该属性，一些改动会影响JSP的编译能力。

示例

```
<%@page extends="java.util.Date" %>
```

6.2.7 import属性——导入需要用到的包

类似于Java中的import语句，用来向JSP文件中导入需要用到的包。

示例

在Page指令中可多次使用该属性来导入多个包。

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
```

或者通过逗号间隔，来导入多个包。

```
<%@ page import="java.util.*,java.text.*" %>
```

在JSP中已经默认导入了以下包：

```
java.lang.*
javax.servlet.*
```


PDF电子书基地 http://dayo1982.400gb.com

```
javax.servlet.jsp.*
javax.servlet.http.*
```

所以，即使没有通过 import 属性设置，在 JSP 页面中也可以调用上述的包中的类。

若要在页面中使用编写的 JavaBeans，也可通过 import 属性来导入。还可以通过<jsp:useBean>动作标签来创建一个 JavaBean 实例来进行调用。

示例

```
<%@page isThreadSafe="true" %>
```

6.2.8 info 属性——设置页面相关信息

该属性可设置为任意字符串，如当前页面的作者或其他有关的页面信息。可通过 Servlet.getServletInfo()方法来获取设置的字符串。

示例

```
<%@ page info="This is index.jsp!" %>
<%=this.getServletInfo()%>
```

访问页面后，将显示以下结果。

```
This is index.jsp!
```

6.2.9 isErrorPage 属性——设置是否使用 exception 异常对象

将该属性值设为 true，此时在当前页面中可以使用 exception 异常对象。若在其他页面中通过 errorPage 属性指定了该页面，则当调用页面出现异常时，则会跳转到该页面，并且在该页面中可通过 exception 对象输出错误信息。相反，如果将该属性设置为 false，则在当前页面中不能使用 exception 对象。该属性默认值为 false。

示例

若当前应用下包含 index.jsp 和 error.jsp 文件。在 index.jsp 页面中进行数据的格式化操作如下：

```
<%@ page contentType="text/html;charset=gb2312" errorPage="error.jsp"%>
<%
    String name="lzw";
    Integer.parseInt(name);          //将字符串转化为整型
%>
```

上述代码将一个非数字格式的字符串转化为整型，因此将发生异常，最终进入 errorPage 属性指定的 error.jsp 页面显示错误信息。

在 error.jsp 页面中需要将 isErrorPage 属性设为 true，然后才能调用 exception 对象输出错误信息。error.jsp 页面的代码如下：

```
<%@ page contentType="text/html;charset=gb2312" isErrorPage="true" %>
```

出现错误，错误如下：

```
<%=exception.getMessage()%>
```

访问 index.jsp 页面后，将显示以下结果。

出现错误！错误如下：

```
For input string: "lzw"
```

6.2.10 isIEIgnored属性——设置是否忽略EL表达式

通过该属性的设置，可以使JSP容器忽略表达式语言“${}”。其值只能为true或false。设为true，则忽略表达式语言。

示例

```
<%@page isELIgnored="false" %>
```

6.2.11 isThreadSafe属性——设置是否以多线程方式处理用户请求

该属性默认值为true，表示当前JSP页面被转换为Servlet后，会以多线程的方式来处理来自多个用户的请求；如果设为false，则转换后的Servlet会实现SigleThreadModel接口，该Servlet将以单线程的方式来处理用户请求，即其他请求必须等待直到前一个请求被处理结束。

示例

```
<%@page isThreadSafe="true" %>
```

6.2.12 pageEncoding属性——设置字符的编码

用来设置JSP页面字符的编码。默认值为“ISO-8859-1”。

示例

```
<%@page pageEncoding="ISO-8859-1" %>
```

6.2.13 session属性——设置当前页面是否支持session

该属性默认值为true，表示当前页面支持session。设为false表示不支持session。

示例

```
<%@page session="true" %>
```

6.3 taglib指令——标签库指令

通过使用taglib指令，开发者就可以在页面中使用这些基本标记或自定义的标记来完成特殊的功能。

语法

```
<%@ taglib uri="tagURI" prefix="tagPrefix" %>
```

参数说明

uri属性：该属性指定了JSP要在web.xml文件中查找的标签库描述符，该描述符是一个标签描述文件（*.tld）的映射。在tld标签描述文件中定义了该标签库中的各个

PDF电子书基地 http://dayo1982.400gb.com

标签名称，并为每个标签指定一个标签处理类。另外，通过 uri 属性直接指定标签描述文件的路径，而无需在 web.xml 文件中进性配置，同样可以使用指定的标记。

prefix 属性：该属性指定一个在页面中使用由 uri 属性指定的标签库的前缀。前缀命名不能为：

```
jsp、jspx、java、javax、sun、servlet 和 sunw
```

示例

开发者可通过前缀来引用标签库中的标签。

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="name" value="yxq"/>
```

上述代码通过<c:set>标签将“yxq”值赋给了变量“name”。

第 7 章　JSP 的动作标签

7.1　<jsp:fallback>——不支持插件时输出提示信息标签

<jsp:fallback>是<jsp:plugin>的子标识，当使用<jsp:plugin>标识加载 Java 小应用程序或 JavaBean 失败时，可通过<jsp:fallback>标识向用户输出提示信息。

语法

```
<jsp:plugin type="applet" code="com.source.MyApplet.class" codebase=".">
  ...
  <jsp:fallback>这里设置错误提示信息</jsp:fallback>
  ...
</jsp:plugin>
```

示例

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase="./applet">
    <jsp:fallback>加载 Applet 失败！ </jsp:fallback>
</jsp:plugin>
```

7.2　<jsp:forward>——请求转发标签

<jsp:forward>动作标签用来将请求转发给另外一个 JSP 或 Servlet 来处理。当该标签被执行后，当前的页面将不再被执行，而是去执行该标签指定的目标页面或 Servlet。

语法

```
<jsp:forward page="文件路径 | 表示路径的表达式"/>
```

如果转发的目标是一个动态文件，还可以向该文件中传递参数。

语法

```
<jsp:forward page="文件路径 | 表示路径的表达式">
  <jsp:param name="参数名 1" value="值 1"/>
  <jsp:param name="参数名 2" value="值 2"/>
  ...
</jsp:forward>
```

参数说明

page 属性：该属性指定了目标文件的路径。如果该值是以“/”开头，表示在当前应用的根目录下查找文件，否则就在当前路径下查找目标文件。请求被转向到的目标文件必须是内部的资源，即当前应用中的资源。

PDF电子书基地 http://dayo1982.400gb.com

<jsp:param>子标签用来向动态的目标文件中传递参数。

示例

若当前应用为 A，在根目录下的 index.jsp 页面中存在下面的代码用来将请求转发到应用 B 中的 logon.jsp 页面。

```
<jsp:forward page="http://localhost:8080/B/logon.jsp"/>
```

那么将出现下面的错误提示：

```
The requested resource (/http://localhost:8080/B/logon.jsp) is not available
```

仔细观察可以看到，错误提示中的路径前自动加入了一个“/”，这是因为 index.jsp 页面在应用 A 的根目录下，当 forward 标签被执行时，会在该目录下来查找 page 属性指定的目标文件，所以会提示资源不存在的信息。换个说法：“<jsp:forward>动作标签只能用于站内转发”。

注意

这里重点提示一下，<jsp:forward>标签实现的是请求的转发操作，而不是请求重定向。它们之间的一个区别就是：进行请求转发时，存储在 request 对象中的信息会被保留并被带到目标页面中；而请求重定向是重新生成一个 request 请求，然后将该请求重定向到指定的 URL，所以事先存储在 request 对象中的信息都不存在了。

7.3 <jsp:getProperty>——获取属性值标签

<jsp:getProperty>属性用来从指定的 Bean 中读取指定的属性值，并输出到页面中。该 Bean 必须具有 getXXX()方法。

语法

```
<jsp:getProperty name="Bean 实例名" property="propertyName"/>
```

参数说明

name 属性：用来指定一个存在某 JSP 范围中的 Bean 实例。<jsp:getProperty>标签将会按照 page、request、session 和 application 的顺序来查找这个 Bean 实例，直到第一个实例被找到。若任何范围内不存在这个 Bean 实例则会抛出异常：“Attempted a bean operation on a null object”。

property 属性：该属性指定要获取属性值的名称，也就是由 name 属性指定的 Bean 中的属性名称。若它指定的值为“userName”，那么 Bean 中必须存在 getUserName()方法，否则会抛出异常：“Cannot find any information on property 'userName' in a bean of type '此处为类名'”。

如果指定 Bean 中的属性是一个对象，那么该对象的 toString()方法将被调用，并输出执行结果。

示例

```
<jsp:useBean id="date" class="java.util.Date" scope="session"/>
<jsp:getProperty name="date" property="month"/>
```

7.4 <jsp:include>——包含文件标签

该动作标签用于向当前的页面中包含其他的文件，这个文件可以是动态文件也可以是静态文件。

语法

```
<jsp:include page="被包含文件的路径" flush="true|false"/>
```

或者向被包含的动态页面中传递参数：

```
<jsp:include page="被包含文件的路径" flush="true|false">
  <jsp:param name="参数名称" valude="参数值"/>
</jsp:include>
```

参数说明

page 属性：该属性指定了被包含文件的路径，其值可以是一个代表了相对路径的表达式。当路径是以“/”开头时，则按照当前应用的根目录查找这个文件；如果路径是以文件名或目录名称开头，那么将按照当前的路径来查找被包含的文件。

flush 属性：表示当输出缓冲区满时，是否清空缓冲区。该属性值为 boolean 型，默认值为 false，通常情况下设为 true。但在 JSP 1.2 以前，flush 属性必须设为 true。

<jsp:param>子标签可以向被包含的动态页面中传递参数。

<jsp:include>标签对包含的动态文件和静态文件的处理方式是不同的。如果被包含的是静态的文件，则页面执行后，在使用了该标签的位置处将会输出这个文件的内容。如果<jsp:include>标签包含的是一个动态的文件，那么 JSP 编译器将编译并执行这个文件。不能通过文件的名称来判断该文件是静态的还是动态的，<jsp:include>标签会识别出文件的类型。

示例

```
<%@ page contentType="text/html;charset=gb2312" %>
<jsp:include page="top.jsp"/>
```

7.5 <jsp:param>——传递单个参数标签

<jsp:param>标签可用来传递参数，被传递的参数以“参数名=值”的形式加入到了请求中。该标签不能单独存在，通常情况下与<jsp:include>、<jsp:forward>和<jsp:plugin>一起使用。

语法

```
<jsp:param name="参数名" value="值"/>
```


PDF电子书基地 http://dayo1982.400gb.com

参数说明：

name 属性：该属性指定了参数的名称，它不支持任何的表达式，其值必须为字符串型。

value 属性：该属性指定了参数的值，它可以是一个代表了具体值的表达式。

示例

如果向目标页面中传递多个参数，可应用多个<jsp:param>标签来实现，如下面的用法：

```
<% String arg1="A"%>
<% String arg2="B"%>
<jsp:forward page="/result.jsp">
   <jsp:param name="select1" value="<%=arg1%>"/>
   <jsp:param name="select2" value="<%=arg2%>"/>
</jsp:forward>
```

在目标页面中可通过调用 request 对象的 getParameter()方法获取传递的参数。

7.6 <jsp:params>——传递多个参数标签

在该标识内可包含多个<jsp:param>子标识，每个<jsp:param>标识指定一个想要加载的 Java Applet 或 Bean 中传递的参数。

语法

```
...
    <jsp:params>
        <jsp:param>标签设置
        <jsp:param>标签设置
    </jsp:params>
...
```

示例

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase="./applet">
    <jsp:params>
        <jsp:param name="username" value="YXQ"/>
        <jsp:param name="userpwd" value="123"/>
    </jsp:params>
</jsp:plugin>
```

7.7 <jsp:plugin>——插入 Applet 或 JavaBean 标签

使用<jsp:plugin>标识可以在页面中插入 Java Applet 小程序或 JavaBean，它们能够在客户端运行。该标识会根据客户端浏览器的版本转换成<object>或<embed>HTML 元素。

语法

```
<jsp:plugin
        type="applet | bean"
```

```
        code=""
        codebase=""
        [name=""]
        [archive=""]
        [align=""]
        [height=""]
        [width=""]
        [hspace=""]
        [vspace=""]
        [jreversion=""]
        [nspluginurl=""]
        [iepluginurl=""]
        [<jsp:params>
            <jsp:param name="parameterName" value="{parameterValue | <%=expression %>}"/>
          </jsp:params>]
        [<jsp:fallback>加载失败提示信息</jsp:fallback>]>
</jsp:plugin>
```

下面对<jsp:plugin>标识中各属性作简要说明。

<jsp:plugin>标识中各属性的简要说明

属　性	说　明
type	该属性指定了所要加载的插件对象的类型，可选值为“bean”和“applet”
code	指定了要加载的 Java 类文件的名称。该名称可包含扩展名和类包名，如“com.applet.MyApplet.class”
codebase	默认值为当前访问的 JSP 页面的路径，该属性用来指定 code 属性指定的 Java 类文件所在的路径
name	指定了加载的 Applet 或 Bean 的名称
archive	指定预先加载的存档文件的路径，多个路径可用逗号进行分隔
align	加载的插件对象在页面中显示时的对齐方式。可选值为“bottom”、“top”、“middle”、“left”和“right”
height 和 width	加载的插件对象在页面中显示时的高度和宽度，单位为像素。这两个属性值支持 JSP 表达式或 EL 表达式
hspace 和 vspace	加载的 Applet 或 Bean 在屏幕或单元格中所留出的空间大小，hspace 表示左右，vspace 表示上下，它们不支持任何表达式
jreversion	在浏览器中执行 Applet 或 Bean 时所需的 Java Runtime Environment(JRE)的版本，默认值为 1.1
nspluginurl 和 iepluginurl	分别指定了 Netscape Navigator 用户和 Internet Explorer 用户能够使用的 JRE 的下载地址
<jsp:params>	在该标识中可包含多个<jsp:param>标识，用来向 Applet 或 Bean 中传递参数
<jsp:fallback>	当加载 Java 类文件失败时，用来显示给用户提示信息

示例

如果 codebase 属性值为“/”或长度为 0 的字符串，那么容器将按照“协议+主机+code 属性值”的路径来查找插件对象。

PDF电子书基地 http://dayo1982.400gb.com

例如：若当前 Web 应用为 JSP_Plugin，index.jsp 文件位于其根目录下。index.jsp 文件包含如下代码：

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase="/">
  <jsp:fallback>加载 Java Applet 小程序失败!</jsp:fallback>
</jsp:plugin>
```

容器将会按照下面的路径来查找 MyApplet.class 文件：

```
http://localhost:8080/com/applet/MyApplet.class
```

示例

如果 codebase 属性值为“.”，那么容器将按照当前访问的 JSP 文件的目录为基础路径开始查找插件对象。查找的路径为“协议+主机+当前访问的 JSP 文件目录+code 属性值指定的路径”。

例如：若当前 Web 应用为 JSP_Plugin，index.jsp 文件位于其根目录下。index.jsp 文件包含如下代码：

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase=".">
  <jsp:fallback>加载 Java Applet 小程序失败!</jsp:fallback>
</jsp:plugin>
```

容器将会按照下面的路径来查找 MyApplet.class 文件：

```
http://localhost:8080/JSP_Plugin/com/applet/MyApplet.class
```

示例

如果 codebase 属性值以“./”开头，那么容器将按照当前访问的 JSP 页面所在的目录加上 codebase 属性指定的目录为基础路径开始查找插件对象。

例如：若当前 Web 应用为 JSP_Plugin，其根目录下存在 index.jsp 文件和 applet 子目录。

在 applet 目录下包含如下子目录和文件：

```
com/applet/MyApplet.class
```

index.jsp 文件中包含以下代码：

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase="./applet">
  <jsp:fallback>加载 Java Applet 小程序失败!</jsp:fallback>
</jsp:plugin>
```

访问 index.jsp 页面后，容器将按照下面的路径来查找 MyApplet.class 文件：

```
http://localhost:8080/JSP_Plugin/applet/com/applet/MyApplet.class
```

所以 MyApplet.class 类将被找到，最终会将执行结果显示在浏览器中。

如果 codebase 属性是以“../”开头，那么容器将按照当前访问的 JSP 页面所在目录的上一级目录加上 codebase 属性指定的目录为基础路径开始查找插件对象。

示例

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class" codebase="./applet" iepluginu rl=
"http://localhost:8080">
```

```
    <jsp:fallback>加载 Java Applet 小程序失败!</jsp:fallback>
</jsp:plugin>
```

若当前的 Internet Explorer 用户没有安装 JRE，则访问包含这段代码的 JSP 页面后将会出现如图 7.1 所示的提示。

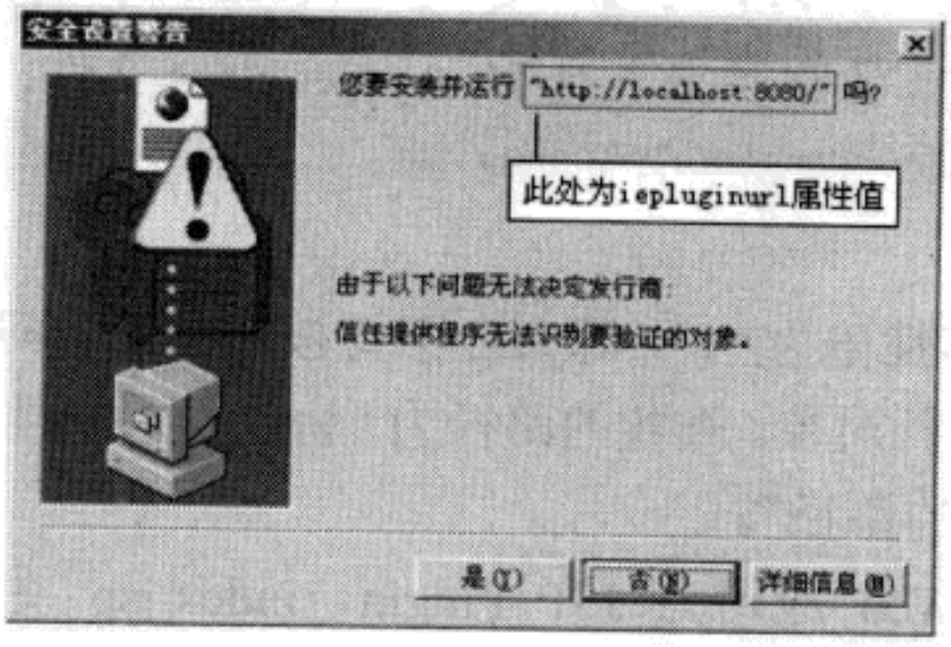


图 7.1　安全设置提示

弹出该提示的前提是需要在浏览器中进行相应的安全设置。打开浏览器中的“工具”→“Internet 选项”子菜单，然后选择“安全”选项卡并单击“自定义级别”按钮，在弹出的“安全设置”对话框中进行如图 7.2 所示的设置。

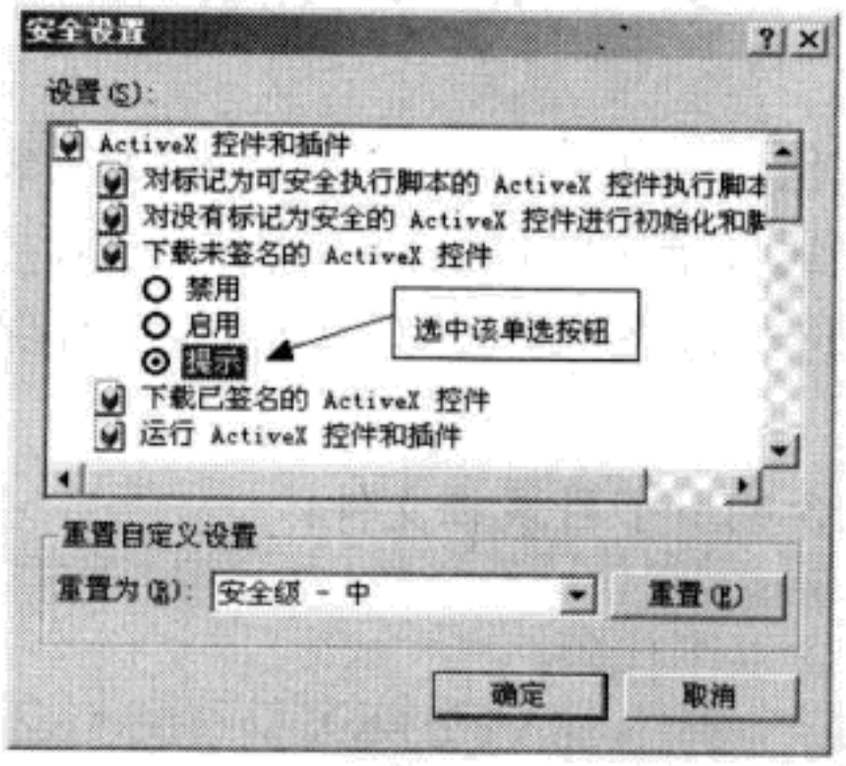


图 7.2　设置浏览器的安全设置

7.8　<jsp:setProperty>——设置属性标签

<jsp:setProperty>标签通常情况下与<jsp:useBean>标签一起使用，它将调用 Bean 中的 setXXX()方法将请求中的参数赋值给由<jsp:useBean>标签创建的 JavaBean 中的对应的简单属性或索引属性。

语法

```
<jsp:setProperty
    name="Bean 实例名"
```


PDF电子书基地 http://dayo1982.400gb.com

```
    {
        property="*" |
        property="propertyName" |
        property="propertyName" param="parameterName" |
        property="propertyName" value="值"
    }/>
```

下面对<jsp:setProperty>标签中各属性作简要说明。

<jsp:setProperty>标签中各属性的简要说明

属　性	说　明
name	该属性是必须存在的属性，用来指定一个 Bean 实例
property	必须存在的属性，可选值为“*”或指定 Bean 中的属性。当取值为“*”时，则 request 请求中的所有参数的值将被一一赋给 Bean 中与参数具有相同名字的属性；若取值为 Bean 中的属性，则只会将 request 请求中与该属性同名的一个参数的值赋给这个 Bean 属性，若此时指定了 param 属性，那么请求中参数的名称与 Bean 属性名可以不同
param	该属性用于指定请求中的参数，通过该属性指定的参数，其值将被赋给由 property 属性指定的 Bean 属性
value	该属性用来指定一个值，它可以是表示具体值的表达式。通常与 property 属性一起使用，表示将指定的值赋给指定的 Bean 属性。value 属性不能与 param 属性一起使用

property 属性取值为“*”时，则 request 请求中的所有参数的值将被一一赋给 Bean 中与参数具有相同名字的属性。如果请求中存在空值的参数，那么 Bean 中对应的属性将不会被赋值为 Null；如果 Bean 中存在一个属性，但请求中没有与之对应的参数，那么该属性同样不会被赋值为 Null。在这两种情况下的 Bean 属性都会保留原来或默认的值。

该种使用方法要求请求中参数的名称和类型必须与 Bean 中属性的名称和类型一致。但由于通过表单传递的参数都是 String 类型的，所以 JSP 会自动将这些参数转换为 Bean 中对应属性的类型。下表列出了 JSP 自动将 String 类型转换为其他类型时所调用的方法。

将 String 类型转换为其他类型的方法

其 他 类 型	转 换 方 法
boolean	java.lang.Boolean.valueOf(String).booleanValue()
Boolean	java.lang.Boolean.valueOf(String)
byte	java.lang.Byte.valueOf(String).byteValue()
Byte	java.lang.Byte.valueOf(String)
double	java.lang.Double.valueOf(String).doubleValue()
Double	java.lang.Double.valueOf(String)
int	java.lang.Integer.valueOf(String).intValue()

续表

其他类型	转换方法
Integer	java.lang.Integer.valueOf(String)
float	java.lang.Float.valueOf(String).floatValue();
Float	java.lang.Float.valueOf(String)
long	java.lang.Long.valueOf(String).longValue()
Long	java.lang.Long.valueOf(String)

示例

不存在 Body 的格式。

```
<jsp:useBean id="logonimg" class="com.javabean.Image" scope="request"/>
<jsp:setProperty name="logonimg" property="x" param="logon.x"/>
<jsp:setProperty name="logonimg" property="y" param="logon.y"/>
```

示例

在 Body 内写入内容的格式。

```
<jsp:useBean id="logonimg" class="com.javabean.Image" scope="request">
    <jsp:setProperty name="logonimg" property="x" param="logon.x"/>
    <jsp:setProperty name="logonimg" property="y" param="logon.y"/>
</jsp:useBean>
```

示例

```
<jsp:useBean id="变量名" scope="JSP 范围">                    //标签开始
  <jsp:setProperty name="变量名" property="*"/>
</jsp:useBean>                                                //标签结束
```

7.9 <jsp:useBean>——创建 Bean 实例标签

通过应用该动作标签可以在 JSP 页面中创建一个 Bean 实例，并且通过属性的设置可以将该实例存储到 JSP 中的指定范围内。如果在指定的范围内已经存在了指定的 Bean 实例，那么将使用这个实例，而不会重新创建。通过<jsp:useBan>标签创建的 Bean 实例可以在 Scriptlet 中应用。

语法

```
<jsp:useBean
            id="变量名"
            scope="page|request|session|application"
            {
              class="package.className"|
              type="数据类型"|
              class="package.className" type="数据类型"|
              beanName="package.className" type="数据类型"
            }/>
<jsp:setProperty name="变量名" property="*"/>
```


PDF电子书基地 http://dayo1982.400gb.com

也可以在标签体内嵌入子标签或其他内容。

语法

```
<jsp:useBean id="变量名" scope="page|request|session|application" .……>
  <jsp:setProperty name="变量名" property="*"/>
</jsp:useBean>
```

下面通过表格对<jsp:useBean>标签中各属性的用法作简要说明。

<jsp:useBean>标签中各属性的简要说明

属　性	说　明
Id	定义一个变量名，程序中将使用该变量名对所创建的 Bean 实例进行引用
Type	指定了 id 属性所定义变量的类型
scope	定位 Bean 实例的范围，缺省值为“page”，其他可选值为 rquest、session 和 application
class	指定一个完整的类名，与 beanName 属性不能同时存在，若没有设置 type 属性，那么必须必须设置 class 属性
beanName	指定一个完整的类名，与 class 属性不能同时存在，设置该属性时必须设置 type 属性，其属性值可以是一个表示完整类名的表达式

示例

通常情况下应用<jsp:useBean>标签的如下格式：

```
<jsp:useBean id="变量名" class="package.className"/>
```

示例

使用 class 属性定位一个类的使用格式如下：

```
<jsp:useBean id="us" class="com.Bean.UserInfo" scope="session"/>
```

示例

class 属性与 type 属性可以指定同一个类，在<jsp:useBean>标签中 class 属性与 type 属性一起使用时的格式如下：

```
<jsp:useBean id="us" class="com.Bean.UserInfo"type="com.Bean.UserBase"scope= "session"/>
```

示例

beanName 属性与 type 属性可以指定同一个类，在<jsp:useBean>标签中 beanName 属性与 type 属性一起使用时的格式如下：

```
<jsp:useBean id="us" beanName="com.Bean.UserInfo" type="com.Bean.UserBase"/>
```

第8章　JSP内置对象

8.1　application对象——应用对象

application对象用于保存所有应用系统中的公有数据，服务器启动并且自动创建application对象以后，只要没有关闭服务器，application对象将一直存在，所有用户可以共享application对象。

application对象与session对象有所区别，session对象和用户会话相关，不同用户的session是完全不同的对象，而用户的application对象都是相同的一个对象，即共享这个内置的application对象。

8.1.1　getAttribute(String name)方法——获取属性值

该方法用于获取由name指定名字的application对象的属性的值。

方法声明

```
Object getAttribute(String name)
```

入口参数：name是要获取的属性的名称。

返回值：指定属性的属性值。

示例

```
<%
application.getAttribute("user");
%>
```

8.1.2　getAttributeName方法——获取属性的初始值

该方法用于获取由name指定的名字的application对象的某个属性的初始值。

方法声明

```
Enumeration getAttributeNames()
```

返回值：保存application对象的所有属性名称的枚举对象。

示例

```
<%
    Enumeration names = application.getAttributeNames();
    while (names.hasMoreElements()) {
        out.println(names.nextElement() + "<br>");
    }
%>
```


PDF电子书基地 http://dayo1982.400gb.com

8.1.3 getContextPath 方法——获取 Web 应用的路径

该方法返回当前 Web 应用的路径，路径以“/”开头。

方法声明

```
String getContextPath()
```

返回值：返回当前 Web 应用的路径。

示例

```
<%
String path = application.getContextPath();
out.print(path);
%>
```

8.1.4 getMajorVersion 方法——获取 Servlet API 的最大版本号

返回服务器支持的 Servlet API 的最大版本号。

方法声明

```
int getMajorVersion()
```

返回值：服务器支持的 Servlet API 最大版本号。

示例

```
<%
int i = application. getMajorVersion ();
out.print(i);
%>
```

8.1.5 getMimiType 方法——获取指定文件的 MIME 类型

返回指定文件的 MIME 类型。

方法声明

```
iString getMimeType(String file)
```

入口参数：file 为指定的文件名。

返回值：返回指定文件的 MIME 类型，如果类型未知则返回 null。

示例

本示例通过 appliction 的 getMimeType 方法输出了 photo.jpg 文件的 MIME 类型。

```
<%
out.print(application.getMimeType("photo.jpg"));
%>
```

8.1.6 getMinorVersion 方法——获取服务器支持的 Servlet API 最小版本号

返回服务器支持的 Servlet API 最小版本号。

方法声明

```
int getMinorVersion ()
```

返回值：服务器支持的 Servlet API 最小版本号。

示例

```
<%
int i = application.getMinorVersion ();
out.print(i);
%>
```

8.1.7 getNamedDispatcher 方法——根据指定参数名返回一个 RequestDispatcher 对象

根据指定参数名返回一个 RequestDispatcher 对象。

方法声明

```
RequestDispatcher getNamedDispatcher(String name)
```

入口参数：name 为一个指定的 servlet 名。

返回值：返回一个 RequestDispatcher 对象。

示例

本示例获取了一个 RequestDispatcher 对象，并转发到指定的 servlet。

```
<%
RequestDispatcher rd = application.getNamedDispatcher("test2");
rd.forward(request, response);
%>
```

8.1.8 getRealPath(String path)方法——获取虚拟路径的真实路径

该方法用于获取虚拟路径的真实路径。

方法声明

```
String getRealPath(String path)
```

入口参数：path 是虚拟路径。

返回值：服务器上的真实路径。

示例

```
<%
        String path=application.getRealPath("index.jsp");
        out.println(path);
%>
```

8.1.9 getRequestDispatcher 方法——根据指定路径名返回一个 RequestDispatcher 对象

根据指定路径名返回一个 RequestDispatcher 对象。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
RequestDispatcher getRequestDispatcher(String path)
```

入口参数：path 为路径。

返回值：返回一个 RequestDispatcher 对象。

示例

本示例获取了一个 RequestDispatcher 对象，并转发到指定的 servlet。

```
<%
RequestDispatcher rd = application.getRequestDispatcher("/page.jsp");
rd.forward(request, response);
%>
```

8.1.10 getResource(Stirng path)方法——获取指定资源的 URL 路径

该方法用于获取指定资源的 URL 路径。

方法声明

```
URL getResource(String path) throws MalformedURLException
```

入口参数：path 是资源的路径。

返回值：该资源的 URL 对象。

抛出异常：MalformedURLException。

示例

```
<%
URL url=application.getResource("index.jsp");
out.println(url);
%>
```

8.1.11 getResourceAsStream 方法——获取指定资源的输入流对象

返回指定资源的输入流对象。

方法声明

```
InputStream getResourceAsStream(String path)
```

入口参数：path 为资源路径名。

返回值：返回一个指定资源的输入流对象，如果不存在则返回 null。

示例

本示例通过 application 的 getResourceAsStream 方法获取了 index.jsp 文件的输入流。

```
<%
InputStream in = application.getResourceAsStream("index.jsp");
%>
```

8.1.12 getResourcePaths 方法——获取路径列表

返回装载指定路径列表文件名的 Set 对象，不包含子目录内的文件。

方法声明

```
Set getResourcePaths(String path)
```

入口参数：path 为资源路径名。

返回值：返回一个装载指定路径列表文件名的 Set 对象，如果资源不存在则返回 null。

示例

本示例通过 application 的 getResourcePaths 方法输出了当前 Web 目录下的所有文件，不包含子目录内的文件。

```
<%
Set set = application.getResourcePaths("/");
for(Iterator it = set.iterator(); it.hasNext();){
    out.print(it.next() + "<br>");
}
%>
```

8.1.13 log 方法——写入日志信息到日志文件

把指定信息写入到 Servlet 的日志文件中。

方法声明 1

```
void log(String msg)
```

入口参数：msg 为要写入日志的信息。

示例

本示例通过 application 的 log 方法把指定信息写入到 Servlet 的日志文件中。

```
<%
application.log("specified message");
%>
```

方法声明 2

把栈轨迹以及给出的 Throwable 异常的说明信息写入 Servlet 的日志文件。

```
void log(String message, Throwable throwable)
```

入口参数：

msg：要写入日志的信息。

throwable：Throwable 对象或异常。

PDF电子书基地 http://dayo1982.400gb.com



示例

本示例通过 application 的 log 方法给出的 Throwable 异常的说明信息写入 Servlet 的日志文件。

```
<%
Throwable t = new Throwable("error");
application.log("specified message" , t);
%>
```

8.1.14 getServletInfo 方法——获取 servlet 容器信息

该方法用于获取 servlet 容器的名称和当前版本的信息。

方法声明

```
String getServerInfo()
```

返回值：服务器的版本信息和名称。

示例

```
<%
        String info=application.getServerInfo();
        out.println(info);
%>
```

8.1.15 removeAttribute(String key)方法——移除属性

该方法用于移除 application 对象中的一个属性。

方法声明

```
void removeAttribute(String name)
```

入口参数：name 是要移除的属性的名称。

示例

```
<%
        application.setAttribute("user","lzw");
        application.removeAttribute("user");
%>
```

8.1.16 setAttribute(String name,Object object)方法——设置属性

该方法用于设置由 name 指定名字的 application 对象的属性值 object。

方法声明

```
void setAttribute(String name,   Object object)
```

入口参数：name 是要设置的属性名称。

object 是要设置的属性值，可以是任何对象。

示例

```
<%
    application.setAttribute("user","lzw");
    Date now=new Date();
    application.setAttribute("now",now);
%>
```

8.2 config 对象——配置对象

8.2.1 getInitParameter 方法——获取单个初始参数的值

该方法用于获取名字为 name 的初始参数的值。

方法声明

```
String getInitParameter(String name)
```

入口参数：name 是要获取的参数的名称。

返回值：参数值。

示例

```
<%
String name=config.getInitParameter("fork");
out.println("parameter value is:"+name);
%>
```

8.2.2 getInitParameterNames 方法——获取全部初始参数的名字

该方法用于获取这个 JSP 的所有的初始参数的名字。

方法声明

```
Enumeration getInitParameterNames()
```

返回值：保存 JSP 所有初始参数名称的枚举对象。

示例

```
<%
Enumeration names = config.getInitParameterNames();
while (names.hasMoreElements()) {
    out.println(names.nextElement()+"<br>");
}
%>
```

8.2.3 getServletContext 方法——获取 Servlet 上下文

该方法用于获取 Servlet 上下文。

方法声明

```
ServletContext getServletContext()
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：Servlet 上下文对象。

示例

```
<%
ServletContext sc=config.getServletContext();
%>
```

8.2.4 getServletName 方法——获取 Servlet 的名字

该方法用于获取 Servlet 的名字。

方法声明

```
String getServletName()
```

返回值：Servlet 的名字。

示例

```
<%
String name=config.getServletName();
out.println("Servlet name is:"+name);
%>
```

8.3 exception 对象——异常对象

在程序运行过程中常常发生一些预料不到的问题，称之为异常。exception 指的是 Web 应用程序所能够识别并处理的问题，在 Java 中，利用名为“try/catch”的关键字来处理异常情况，如果在 JSP 页面中出现没有捕捉到的异常，就会生成 exception 对象，并把这个 exception 对象传送到在 page 指令中设定的错误页面中，然后在错误提示页面中处理相应的 exception 对象。

exception 对象只有在错误页面（在页面指令里有 isErrorPage=true 的页面）才可以使用。

8.3.1 fillInStackTrace 方法——在异常堆栈跟踪中填充

在异常堆栈跟踪中填充。

方法声明

```
public Throwable fillInStackTrace()
```

返回值：对此 Throwable 实例的引用。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
    Integer.parseInt("测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
    Throwable t = exception.fillInStackTrace();
    t.printStackTrace();
%>
```

8.3.2　getCause 方法——返回产生异常的原因

返回此 throwable 的 cause。

方法声明

```
public Throwable getCause()
```

返回值：此 throwable 的 cause，如果 cause 不存在或是未知的，则返回 null。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
    Integer.parseInt("测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
    Throwable t = exception. getCause();
    t.printStackTrace();
%>
```

8.3.3　getMessage 方法——获取异常消息文本

该方法获取异常消息文本。

方法声明

```
public String getMessage()
```

返回值：异常对象的消息字符串。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
```


PDF电子书基地 http://dayo1982.400gb.com

```
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
        out.print(exception.getMessage());
%>
```

8.3.4 getLocalizedMessage 方法——获取本地化语言的异常信息

该方法用于获取本地化语言的异常信息。

方法声明

```
public String getLocalizedMessage()
```

返回值：异常对象的本地化消息字符串。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" errorPage="error.jsp" pageEncoding="GBK"%>
<%
        Integer.parseInt("异常测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
        out.print(exception.getLocalizedMessage());
%>
```

8.3.5 getStackTrace 方法——获取堆栈跟踪元素的数组

返回堆栈跟踪元素的数组，每个元素表示一个堆栈帧。

方法声明

```
public StackTraceElement[] getStackTrace()
```

返回值：堆栈跟踪元素的数组，表示与此 throwable 相关的堆栈跟踪。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
StackTraceElement[] stes = exception.getStackTrace();
for(StackTraceElement s : stes){
      out.print(s.toString());
}
%>
```

8.3.6 initCause 方法——设置初始化的异常信息

将此 throwable 的 cause 初始化为指定值。

方法声明

```
public Throwable initCause(Throwable cause)
```

入口参数：Cause 是导致抛出此 Throwable 的 Throwable。

返回值：对此 Throwable 实例的引用。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
      Integer.parseInt("测试");
%>
```

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
exception.initCause(new Throwable("Not Number"));
exception.printStackTrace();
%>
```

8.3.7 printStackTrace 方法——输出异常的栈跟踪轨迹

该方法用于输出异常的栈跟踪轨迹。

方法声明 1

```
public void printStackTrace()
```

示例

在首页面产生异常，在 error.jsp 页面调用 printStackTrace 方法在控制台输出异常的栈跟轨迹。

首先在首页面产生异常。

```
<%@ page language="java" errorPage="error.jsp" pageEncoding="GBK"%>
<%
      Integer.parseInt("异常测试");
%>
```


PDF电子书基地 http://dayo1982.400gb.com

在 error.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
        exception.printStackTrace();
%>
```

方法声明 2

将此 throwable 及其追踪输出到指定的输出流。

```
public void printStackTrace(PrintStream s)
```

入口参数：s 为用于输出的 PrintStream。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面将异常消息输出到硬盘上的指定文件中。

```
<%
PrintStream s = new PrintStream("c:\\servlet.log");
exception.printStackTrace(s);
s.close();
%>
```

方法声明 3

将此 throwable 及其追踪输出到指定的 PrintWriter。

```
public void printStackTrace(PrintWriter s)
```

入口参数：s 为用于输出的 PrintWriter。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面将异常消息输出到硬盘上的指定文件中。

```
<%
PrintWriter s = new PrintWriter("c:\\servlet.log");
exception.printStackTrace(s);
s.close();
%>
```

8.3.8 setStackTrace 方法——设置堆栈跟踪元素

设置将由 getStackTrace()返回，并由 printStackTrace()和相关方法输出的堆栈跟踪元素，此方法用于高级应用。

方法声明

```
public void setStackTrace(StackTraceElement[] stackTrace)
```

入口参数：stackTrace 为要与此 Throwable 关联的堆栈跟踪元素。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面设置堆栈跟踪元素（只是演示用法）。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
StackTraceElement[] stackTrace = exception.getStackTrace();
exception.setStackTrace(stackTrace);
%>
```

8.3.9 toString 方法——返回异常的简短描述

返回此 throwable 的简短描述。

方法声明

```
public String toString()
```

返回值：返回此 throwable 的简短描述。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
        Integer.parseInt("测试");
%>
```

在 error.jsp 页面设置堆栈跟踪元素（只是演示用法）。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
out.print(exception.toString());
%>
```


PDF电子书基地 http://dayo1982.400gb.com

8.4 out 输出对象——输出对象

out 对象主要用来向客户端输出各种数据类型的内容，并且管理应用服务器上的输出缓冲区，缓冲区默认值一般是 8KB，可以通过页面指令 page 来改变默认值。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据。为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

out 对象被封装为 javax.servlet.jsp.JspWriter 类的对象，在实际应用上 out 对象会通过 JSP 容器会变换为 java.io.PrintWriter 类的对象。Servlet 使用的是 java.io.PrintWriter 对象向网页输出数据。

8.4.1 clear 方法——清除缓冲区里的数据

该方法用于清除缓冲区里的数据，但不会把数据输出到客户端，即 JSP 页面。

方法声明

```
public abstract void clear() throws IOException
```

抛出异常：IOException。

示例

```
<%
    out.println("Java 编程词典");
    out.clear ();
%>
```

8.4.2 clearBuffer 方法——清除缓冲区里的数据

该方法用于清除缓冲区里的数据，并把数据输出到客户端，即 JSP 页面。

方法声明

```
public abstract void clearBuffer() throws IOException
```

抛出异常：IOException。

示例

```
<%
    out.println("Java 编程词典");
    out.clearBuffer();
%>
```

8.4.3 close 方法——关闭 JSP 的输出流

该方法用于关闭 JSP 的输出流。执行该方法之后，无法在向 JSP 页面输出任何内容。

方法声明

```
public abstract void close() throws IOException
```

抛出异常：IOException。

示例

```
<%
    out.println("Java 编程词典");
    out.close();
    out.println("JSP 页面");
%>
```

该示例的输出结果为“Java 编程词典”，并且抛出 IOException 异常，因为输出流已经关闭了。

8.4.4 flush 方法——清空并输出当前缓冲区

该方法用于清空当前缓冲区，如果缓冲区中有数据，则输出缓冲里的数据。

方法声明

```
public abstract void flush() throws IOException
```

抛出异常：IOException。

示例

```
<%
    out.println("Java 编程词典");
    out.flush();
%>
```

8.4.5 getBufferSize 方法——获得输出流缓冲区的大小

该方法用于获得输出流缓冲区的大小。

方法声明

```
public int getBufferSize()
```

返回值：JSP 页面缓冲区的大小，单位为字节。返回值为整数类型。

示例

```
<%
    int size = out.getBufferSize();
    out.println("JSP 页面的缓冲区大小为：" + size);
%>
```

该示例在 JSP 页面的输出结果为：“JSP 页面的缓冲区大小为：8192”。

8.4.6 getRemaining 方法——获得缓冲区的可用空间

该方法用于获得缓冲区中没有被占用的空间的大小。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public abstract int getRemaining()
```

返回值：当前缓冲区中剩余的空间大小。

示例

```
<%
    int size = out.getRemaining();
    out.println("JSP 页面的剩余缓冲区大小为：" + size);
%>
```

该实例在 JSP 页面输出的结果是：“JSP 页面的剩余缓冲区大小为：7944”。

8.4.7 isAutoFlush 方法——判断缓冲区是否自动刷新

该方法用于判断缓冲区是否自动刷新。

方法声明

```
public boolean isAutoFlush()
```

返回值，如果 AutoFlush 为真，则返回 True；反之，则返回 False。

示例

```
<%
    int size = out.getRemaining();
    out.println("JSP 页面是否自动刷新：" + out.isAutoFlush());
%>
```

该实例在 JSP 页面的输出结果为：“JSP 页面是否自动刷新：true”。

8.4.8 newLine 方法——写入一个行分隔符

写入一个行分隔符。行分隔符字符串由系统属性 line.separator 定义，并且不一定是单个新行('\n')符。

方法声明

```
public abstract void newLine() throws IOException
```

抛出异常：IOException。

示例

```
<%
out.print("明日科技");
out.newLine();
out.print("编程词典");
%>
```

8.4.9 print 方法——输出数据到 JSP 页面

print 方法用于输出数据到 JSP 页面。

方法声明 1

该方法用于向 JSP 页面输出 Boolean 类型的布尔值。输出的内容将显示在 JSP 页面上。

```
public abstract void print(boolean b) throws IOException
```

入口参数：b 是要输出的布尔表达式、变量或布尔常量。

返 回 值：无。

示例

在 JSP 页面输出条件判断的表达式结果。

```
<%
        out.print(3>2);
%>
```

运行结果将在页面输出“true”。

方法声明 2

该方法用于向 JSP 页面输出 char 类型的字符。输出的内容将显示在 JSP 页面上。

```
public abstract void print(char c) throws IOException
```

入口参数：c 是要输出的字符运算表达式、变量或字符常量。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出字符。

```
<%
        char c='A';
        out.print(c);
%>
```

该示例将在 JSP 页面输出字符“A”。

方法声明 3

该方法用于向 JSP 页面输出 int 类型的整数。输出的整数将显示在 JSP 页面上。

```
public abstract void print(int i) throws IOException
```

入口参数：i 是要输出的整数运算表达式、变量或整数常量。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出整数。

PDF电子书基地 http://dayo1982.400gb.com

```
<%
        int num=96;
        out.print(num);
%>
```

该示例将在 JSP 页面输出整数“96”。

方法声明 4

该方法用于向 JSP 页面输出 long 类型的长整数。输出的整数将显示在 JSP 页面上。

```
public abstract void print(long l)   throws IOException
```

入口参数：l 是要输出的整数运算表达式、变量、整数常量或长整数常量。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出整数。

```
<%
        int num=96;
        long sum=10;
        out.print(sum+num);
%>
```

该示例将在 JSP 页面输出“106”，int 类型和 long 类型相加，运算结果将自动转换为 long 类型。

方法声明 5

该方法用于向 JSP 页面输出 float 类型的单精度浮点数。输出的浮点数将显示在 JSP 页面上。

```
public abstract void print(float f)   throws IOException
```

入口参数：f 是要输出的浮点数运算表达式、float 类型常量或变量。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出浮点数。

```
<%
        int num=96;
        float price=0.76f;
        out.print(num*price);
%>
```

该示例将在 JSP 页面输出“72.96”，int 类型和 float 类型进行算数运算，运算结果将自动转换为浮点类型。

方法声明 6

该方法用于向 JSP 页面输出 double 类型的双精度浮点数。输出的内容将显示在 JSP 页面上。

```
public abstract void print(double d)  throws IOException
```

入口参数：d 是要输出的浮点数运算表达式、double 类型常量或变量。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出 double 类型的浮点数。

```
<%
        int num=96;
        double price=1.76f;
        out.print(num*price);
%>
```

该示例将在 JSP 页面输出双精度浮点数“168.95999908447266”，int 类型和 double 类型进行算数运算，运算结果将自动转换为 double 类型。

方法声明 7

该方法用于向 JSP 页面输出 char 类型的字符数组。输出的内容将显示在 JSP 页面上。

```
public abstract void print(char[] s)  throws IOException
```

入口参数：s 是要输出的字符数组。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出字符数组组成的字符串。

```
<%
        char[] s=new char[]{'J','a','v','a','编','程','词','典'};
        out.print(s);
%>
```

该示例将在 JSP 页面输出字符串“Java 编程词典”。

方法声明 8

该方法用于向 JSP 页面输出 String 类型的字符串。输出的内容将显示在 JSP 页面上。

```
public abstract void print(String s)  throws IOException
```

入口参数：s 是要输出的字符串对象。

返 回 值：无。

抛出异常：IOException。

PDF电子书基地 http://dayo1982.400gb.com

示例

在 JSP 页面输出 String 类型的字符串。

```
<%
        String s="Java 编程词典，是编程人员的宝典";
        out.print(s);
%>
```

该示例将在 JSP 页面输出字符串“Java 编程词典，是编程人员的宝典”。

方法声明 9

该方法用于向 JSP 页面输出任意类型的 Java 对象所转换的字符串。输出的内容将显示在 JSP 页面上。

```
public abstract void print(Object obj)    throws IOException
```

入口参数：obj 是要输出的 Java 对象。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出 Date 类的实例对象。

```
<%
        Date date=new Date();
        out.print(date);
%>
```

该示例将在 JSP 页面输出字符串“Mon Oct 27 08:29:37 CST 2008”。

8.4.10 println 方法——向 JSP 页面输出空字符串和换行符号

该方法用于向 JSP 页面输出空字符串和换行符号。

方法声明

```
public abstract void println()    throws IOException
```

入口参数：无。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出空字符串和换行符。

```
<%
        out.println();
%>
```

8.4.11 其他 println 方法

该方法和 print 方法一样，也拥有多个重载方法（就是支持不同参数的方法），但是该方法在其他 print 方法的基础上增加了'\n'换行符，也就是在原有内容输出

之后，会输出换行字符。但是这个换行字符只影响 JSP 生成的页面源代码，它在 JSP 页面中不会起到换行的作用，因为 JSP 页面只能使用 HTML 语言的
标签实现换行。

方法声明

```
public abstract void println(boolean b)    throws IOException
public abstract void println(char c)    throws IOException
public abstract void println(int i)    throws IOException
public abstract void println(long l)    throws IOException
public abstract void println(float f)    throws IOException
public abstract void println(double d)    throws IOException
public abstract void println(char[] cs)    throws IOException
public abstract void println(String s)    throws IOException
public abstract void println(Object obj)    throws IOException
```

入口参数：

b：要输出的布尔表达式、变量或布尔常量。

c：要输出的字符运算表达式、变量或字符常量。

i：要输出的整数运算表达式、变量或整数常量。

l：要输出的整数运算表达式、变量、整数常量或长整数常量。

f：要输出的浮点数运算表达式、float 类型常量或变量。

d：要输出的浮点数运算表达式、double 类型常量或变量。

cs：要输出的字符数组。

s：要输出的字符串对象。

obj：要输出的 Java 对象。

返 回 值：无。

抛出异常：IOException。

示例

在 JSP 页面输出 Date 类的实例对象。

```
<%
        out.println(false);
        out.println('A');
        int i = 10;
        out.println(i);
        long l = 300;
        out.println(l);
        float f = 1.889f;
        out.println(f);
        double d = 3422.332;
        out.println(d);
        out.println(new char[] { 'a', 'b', 'c' });
```


PDF电子书基地 http://dayo1982.400gb.com

```
        out.println("Java 编程词典");
        Date date = new Date();
        out.println(date);
    %>
```

8.5 page 对象——页面对象

Page 对象是为了执行当前页面的应答请求而设置的 Servlet 类的实体（Instance），即显示的是 JSP 页面自身，只有在 JSP 页面内才是合法的。它是 java.lang.Object 类的一个实例，在脚本语言为 Java 的时候，Page 对象只是 this 关键字的一个代名词，它可以调用当前 JSP 页面定义的任何方法。该对象不经常使用。

8.5.1 equals 方法——比较两个对象是否相等

指示其他某个对象是否与此对象“相等”。

方法声明

```
public boolean equals(Object obj)
```

入口参数：obj 为要与之比较的引用对象。

返回值：如果此对象与 obj 参数相同，则返回 true；否则返回 false。

示例

```
<%
    boolean b = page.equals(page);
    out.print(b);
%>
```

8.5.2 getClass 方法——获取 page 对象

该方法用于获得 page 对象也就是当前 JSP 页面的 Servlet 类。

方法声明

```
public final Class<?> getClass()
```

返回值：返回此 JSP 页面的运行时类，也就是 JSP 页面编译后的 Servlet 类。

示例

```
<%
    out.println("当前 JSP 页面的类是：" + page.getClass());
%>
```

该实例将在 JSP 页面输出：“当前 JSP 页面的类是：class org.apache.jsp.fds_jsp”。

8.5.3 hashCode 方法——返回对象的哈希码值

返回该对象的哈希码值。

方法声明

```
public int hashCode()
```

返回值：返回该对象的哈希码值。

示例

输出该对象的哈希码值。

```
<%
    out.print(page.hashCode());
%>
```

8.6　pageContext 对象——页面上下文对象

pageContext 对象提供存取所有关于 JSP 程序执行时期所要用到的属性方法，它被封装成 javax.servlet.jsp.PageContext 类，为 JSP 页面的上下文，管理对属于 JSP 中可见部分中已命名对象的访问，它的创建和初始化都是由容器来完成的，JSP 页面里可以直接使用 pageContext 对象的句柄。

8.6.1　forward 方法——页面转发

该方法可以把页面转发到另一个页面或者 servlet 组件上。

方法声明

```
public abstract void forward(String relativeUrlPath)
                        throws ServletException,
                               IOException
```

入口参数：relativeUrlPath 是目标的相对地址。

示例

```
<%
    pageContext.forward("count.jsp");
%>
```

8.6.2　getErrorData 方法——获取错误信息

该方法返回一个包含错误信息的 ErrorData 对象。

方法声明

```
public ErrorData getErrorData()
```

返回值：返回一个包含错误信息的 ErrorData 对象，如果不在错误页调用，返回的信息是无意义的。

示例

```
<%
    ErrorData error = pageContext.getErrorData();
%>
```


PDF电子书基地 http://dayo1982.400gb.com

8.6.3 getAttribute 方法——在指定范围内检索属性

该方法用来检索一个特定的已命名的对象的范围。

方法声明

```
Object getAttribute(String name,int scope)
```

入口参数：

name：要获取属性的名称。

Scope：要搜索属性的范围。

返回值：指定范围中的属性值。

scope 参数的取值如下表所示。

scope 参数的取值范围

常　数	说　明
APPLICATION_SCOPE	在 application 对象范围中搜索
PAGE_SCOPE	在 page 对象范围中搜索
REQUEST_SCOPE	在 request 对象范围中搜索
SESSION_SCOPE	在 session 对象范围中搜索

示例

在 session 对象范围中搜索 user 属性。

```
<%
    pageContext.getAttribute("user",pageContext.SESSION_SCOPE);
%>
```

8.6.4 getException 方法——获取异常信息

该方法用于获取当前的 Exception 对象，不过当前页面要设置为 error page。

方法声明

```
public abstract Exception getException()
```

返回值：当前页面的 Exception 异常对象。

示例

在首页面产生异常，然后在 error.jsp 页面获取异常，并显示异常信息。

```
<%@ page language="java" errorPage="error.jsp" pageEncoding="GBK"%>
<%
    Integer.parseInt("d");
%>
```

编写 error.jsp 页面。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true"%>
<%
```

```
        Exception e = pageContext.getException();
        out.print(e.getMessage());
    %>
```

8.6.5 getPage 方法——获取当前 page 对象

该方法返回当前 page 对象。

方法声明

```
public abstract Object getPage()
```

返回值：返回 page 对象。

示例

```
<%
page = pageContext.getPage();
%>
```

8.6.6 getRequest 方法——获取当前的 request 对象

该方法用于获取当前的 request 对象。

方法声明

```
public abstract ServletRequest getRequest()
```

返回值：返回当前页面的 request 对象。

示例

```
<%
        HttpServletRequest req= (HttpServletRequest)pageContext.getRequest();
        String name=req.getParameter("name");
%>
```

8.6.7 getResponse 方法——获取当前的 response 对象

该方法用于获取当前的 response 对象。

方法声明

```
public abstract ServletResponse getResponse()
```

返回值：当前页面的 response 对象。

示例

```
<%
        HttpServletResponse res= (HttpServletResponse)pageContext.getResponse();
        res.setHeader("refresh","1");
        out.print(new Date());
%>
```

8.6.8 getServletConfig 方法——获取当前页面的 ServletConfig 对象

该方法用于获取当前页面的 ServletConfig 对象。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public abstract ServletConfig getServletConfig()
```

返回值：当前页面的 ServletConfig 对象。

示例

获取 ServletConfig 对象，然后获取 Servlet 的名字，输出到页面。

```
<%
    ServletConfig conf = pageContext.getServletConfig();
    String servletName=conf.getServletName();
    out.println("name="+servletName);
%>
```

8.6.9 getServletContext 方法——获取 ServletContext 的实例

该方法返回 ServletContext 的实例。

方法声明

```
public abstract ServletContext getServletContext()
```

返回值：返回 ServletContext 的实例。

示例

```
<%
ServletContext sc = pageContext.getServletContext();
%>
```

8.6.10 getSession 方法——获取当前的 session 对象

该方法返回当前的 session 对象。

方法声明

```
public abstract HttpSession getSession()
```

返回值：返回当前的 session 对象。

示例

本示例通过 pageContext 对象的 getSession 得到当前 session。

```
<%
HttpSession s = pageContext.getSession();
s.setAttribute("user", "明日科技");
%>
```

8.6.11 handlePageException 方法——抛出异常

抛出异常，不进行处理。

方法声明 1

```
' public abstract void handlePageException(Exception e)
        throws ServletException, IOException
```

入口参数：e 为 Exception 对象。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
    Integer.parseInt("测试");
%>
```

在 error.jsp 页面抛出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true" errorPage="error2.jsp"%>
<%
    pageContext.handlePageException(exception);
%>
```

在 error2.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true" %>
<%
out.print(exception.getMessage());
%>
```

方法声明 2

```
public abstract void handlePageException(Throwable t)
    throws ServletException, IOException
```

入口参数：t Throwable 对象。

示例

在首页面产生异常，在 error.jsp 页面显示该异常的消息字符串。

首先在首页面产生异常。

```
<%@ page language="java" pageEncoding="GBK" errorPage="error.jsp"%>
<%
    Integer.parseInt("测试");
%>
```

在 error.jsp 页面抛出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true" errorPage="error2.jsp"%>
<%
    Throwable t = exception.getCause();
    pageContext.handlePageException(t);
%>
```

在 error2.jsp 页面输出异常消息。

```
<%@ page language="java" pageEncoding="GBK" isErrorPage="true" %>
<%
out.print(exception.getMessage());
%>
```


PDF电子书基地 http://dayo1982.400gb.com

8.6.12 include 方法——包含其他文件

向当前的页面中包含其他的文件。

方法声明 1

```
public abstract void include(String relativeUrlPath)
        throws ServletException, IOException
```

入口参数：relativeUrlPath 指定了被包含文件的路径，其值可以是一个代表了相对路径的表达式。当路径是以“/”开头时，则按照当前应用的根目录查找这个文件；如果路径是以文件名或目录名称开头，那么将按照当前的路径来查找被包含的文件。

示例

向当前的页面中包含其他的文件。

```
<%@ page language="java" pageEncoding="GBK"%>
<%
    pageContext.include("/top.jsp");
%>
```

方法声明 2

```
public abstract void include(String relativeUrlPath, boolean flush)
        throws ServletException, IOException
```

入口参数：

relativeUrlPath 指定了被包含文件的路径，其值可以是一个代表了相对路径的表达式。当路径是以“/”开头时，则按照当前应用的根目录查找这个文件；如果路径是以文件名或目录名称开头，那么将按照当前的路径来查找被包含的文件。

flush 表示是否清空缓冲区。

示例

向当前的页面中包含其他的文件。

```
<%@ page language="java" pageEncoding="GBK"%>
<%
    pageContext.include("/top.jsp", true);
%>
```

8.6.13 initialize 方法——初始化 PageContext 对象

初始化 PageContext 对象。

方法声明

```
public abstract void initialize(Servlet servlet,
                                ServletRequest request,
                                ServletResponse response,
                                String errorPageURL,
```

```
                                    boolean needsSession,
                                    int bufferSize,
                                    boolean autoFlush)
                            throws IOException,
                                    IllegalStateException,
                                    IllegalArgumentException
```

入口参数

servlet：当前 Servlet 对象。

request：request 对象。

response：response 对象。

errorPageURL：错误页的 url。

needsSession：session 对象。

bufferSize：缓冲区的大小。

autoFlush：是否自动刷新缓冲区。

示例

本示例通过 pageContext 对象的 initialize 方法进行初始化。

```
<%
pageContext.initialize(this, request, response, "error.jsp", true, 8, true);
%>
```

8.6.14 release 方法——释放相关资源

释放相关资源。

方法声明

```
public abstract void release()
```

示例

本示例通过 pageContext 对象的 getSession 得到当前 session。

```
<%
pageContext. release();
%>
```

8.6.15 removeAttribute 方法——移除指定范围的对象中的属性

该方法用于移除指定范围的对象中的属性。

方法声明

```
public abstract void removeAttribute(String name, int scope)
```

入口参数

name 是要设置的属性名称。

scope 是要设置属性的范围对象。

scope 参数的取值如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

scope 参数的取值范围

常　数	说　明
APPLICATION_SCOPE	在 application 对象范围中搜索
PAGE_SCOPE	在 page 对象范围中搜索
REQUEST_SCOPE	在 request 对象范围中搜索
SESSION_SCOPE	在 session 对象范围中搜索

示例

从 application 对象范围中移除“user”属性。

```
<%
        pageContext.removeAttribute("user",pageContext.APPLICATION_SCOPE);
%>
```

8.6.16　setAttribute 方法——在指定对象范围中设置属性

该方法用于在指定对象范围中设置属性。

方法声明

```
public abstract void setAttribute(String name, Object value, int scope)
```

入口参数

name 是要设置的属性名称。

value 是要设置的属性值。

scope 是要设置属性的范围对象。

scope 参数的取值如下表所示。

scope 参数的取值范围

常　数	说　明
APPLICATION_SCOPE	在 application 对象范围中搜索
PAGE_SCOPE	在 page 对象范围中搜索
REQUEST_SCOPE	在 request 对象范围中搜索
SESSION_SCOPE	在 session 对象范围中搜索

示例

在 application 对象中设置“user”属性。

```
<%
        pageContext.setAttribute("user","lzw",pageContext.APPLICATION_SCOPE);
%>
```

8.7　request 对象——请求对象

request 对象是从客户端向服务器发出请求，可以通过这个对象了解客户端的信息。例如，获得客户端提交的数据、获得客户端 IP 等。该对象继承了 ServletRequest

接口，被包装成 HttpServletRequest 接口。

8.7.1 getAttribute 方法——获取指定的属性值

该方法用于获取指定的属性值，如果指定的属性值不存在，则会返回一个 null 值。

方法声明

```
Object getAttribute(String name)
```

入口参数：name 是要获取属性值的属性名称。

返回值：返回指定的属性值或 null 值。

示例

```
<%
        Object date=request.getAttribute("time");
        out.println("time 属性为："+date);
%>
```

该示例的运行结果将在 JSP 页面输出：“time 属性为：null”，因为没有设置过“time”属性，所以返回值为 null 值。

8.7.2 getAttributeNames 方法——获取 request 对象所有属性的名字集合

返回 request 对象所有属性的名字集合，其结果是一个枚举的实例。

方法声明

```
Enumeration getAttributeNames()
```

返回值：request 对象所设置的所有属性名字的枚举对象。

示例

```
<%
        request.setAttribute("name","lzw");                    // 设置 name 属性
        request.setAttribute("age","27");                      // 设置 age 属性
        request.setAttribute("sex","男");                      // 设置 sex 属性
        Enumeration e=request.getAttributeNames();
        while(e.hasMoreElements()){
                out.println(e.nextElement()+"<br>");
        }
%>
```

该示例将在 JSP 页面输出 request 对象设置的所有属性的名称。

8.7.3 getAuthType 方法——获取保护 servlet 的认证方案名

返回用来保护 servlet 的认证方案名，如：BASIC 或 SSL。

方法声明

```
String getAuthType()
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：返回用来保护 servlet 的认证方案名，如果 servlet 未受保护返回 null。

示例

```
<%
    out.print(request.getAuthType());
%>
```

8.7.4 getCookies 方法——获取所有的 cookie 对象

返回客户端的所有 cookie 对象，结果是一个 cookie 数组。

方法声明

```
Cookie[] getCookies()
```

返回值：Cookie 类型的数组，它包含客户端的所有 Cookie 对象。

示例

```
<%
    Cookie[] cookies = request.getCookies();
    out.println("客户端包含"+cookies.length+"个 cookie");
%>
```

该示例将在 JSP 页面输出：“客户端包含 1 个 cookie”。

注意

该结果将根据不同的客户端，显示不同的结果。

8.7.5 getCharacterEncoding 方法——获取请求页面的字符编码格式

返回请求页面中的字符编码格式。

方法声明

```
String getCharacterEncoding()
```

返回值：返回发出该请求的页面的编码格式的字符串名称。

示例

```
<%
    String encoding=request.getCharacterEncoding();
    out.println("请求页面的编码格式为： "+encoding);
%>
```

8.7.6 getContentLength 方法——获取请求的 Body 的长度

该方法用于获取请求的 Body 的长度，如果不确定长度，返回-1。

方法声明

```
int getContentLength()
```

返回值：请求对象的 Body 长度，返回类型为 int 类型。

示例

```
<%
    int len=request.getContentLength();
    out.println("请求内容的长度为： "+len);
%>
```

8.7.7 getContextPath 方法——获取当前 Web 所在的目录

返回当前 Web 所在的目录。

方法声明

```
String getContextPath()
```

返回值：返回当前 Web 所在的目录。

示例

输出当前 Web 所在的目录。

```
<%
    out.print(request.getContextPath());
%>
```

8.7.8 getDateHeader 方法——获取构建 Date 对象的 long 值

该方法返回指定的头名称的构建 Date 对象的 long 值。

方法声明

```
long getDateHeader(String name)
```

入口参数：name header 的名称。

返回值：返回构建 Date 对象的 long 值。如果不包含 header 返回-1。

示例

```
<%
    out.println(request.getDateHeader("Expires"));
%>
```

8.7.9 getHeader 方法——获得 Http 协议定义的文件头信息

该方法用于获得 Http 协议定义的文件头信息。

方法声明

```
String getHeader(String name)
```

入口参数：name 是文件头的 header 名称。

返回值：header 的取值。

示例

```
<%
    String ct=request.getHeader("content-type");
```


PDF电子书基地 http://dayo1982.400gb.com

```
        out.println("Content-type="+ct);
%>
```

该示例将在 JSP 页面输出：“Content-type=application/x-www-form-urlencoded”。

8.7.10 getHeaders 方法——获取指定请求头的所有值

该方法将返回指定名字的 request Header 的所有值，其结果是一个枚举对象。

方法声明

```
Enumeration getHeaders(String name)
```

入口参数：name 是文件头的 header 名称。

返回值：Enumeration 类的枚举对象，该对象包含指定 header 的所有取值。

示例

```
<%
        Enumeration cts=request.getHeaders("accept-encoding");
        while(cts.hasMoreElements()){
                out.println("accept-encoding="+cts.nextElement()+"<br>");
        }
%>
```

8.7.11 getHeadersNames 方法——获取所有请求头的名字

该方法将返回所有 request Header 的名字，其结果是一个枚举的实例对象。

方法声明

```
Enumeration getHeaderNames()
```

返回值：Enumeration 类的枚举对象，该对象包含所有 request Header 的名字。

示例

```
<%
        Enumeration e=request.getHeaderNames();
        while(e.hasMoreElements()){
                out.println(e.nextElement()+"<br>");
        }
%>
```

该示例将在 JSP 页面输出如下结果：

```
accept
accept-language
ua-cpu
accept-encoding
user-agent
host
connection
cookie
```

8.7.12 getInputStream 方法——获取请求对象的输入流

返回页面 request 请求对象的输入流，用于获得请求中的数据。

方法声明

```
ServletInputStream getInputStream() throws IOException
```

返回值：ServletInputStream 类的示例对象，它是 request 对象的输入流。

抛出异常：IOException。

示例

```
<%
    ServletInputStream is = request.getInputStream();
%>
```

8.7.13 getIntHeader 方法——获取请求头信息

以 int 格式根据名称返回客户请求中对应的头信息（header）。

方法声明

```
int getIntHeader(String name)
```

入口参数：name 为 header 的名称。

返回值：返回构建 Date 对象的 long 值。如果不存在指定的 header 名返回-1。

抛出异常：NumberFormatException，如果不能转换成 int 格式。

示例

```
<%
    out.println(request.getIntHeader("Expires"));
%>
```

8.7.14 getMethod 方法——获取客户端向服务器端传送数据的方法

该方法可以获得客户端向服务器端传送数据的方法，如 get、post、header、trace 等。

方法声明

```
String getMethod()
```

返回值：客户端的表单提交方法的名称。

示例

```
<%
    String mName=request.getMethod();
    out.println("表单提交请求使用的方法是："+mName);
%>
```

该示例运行后，将在 JSP 页面显示“表单提交请求使用的方法是：GET”。

PDF电子书基地 http://dayo1982.400gb.com

8.7.15 getParameter 方法——获取客户端传送的参数值

该方法用于获得客户端传送给服务器端的参数值，该参数是由 name 指定的，通常是表单中的参数。

方法声明

```
String getParameter(String name)
```

入口参数：要获取参数的名称。

返回值：指定名称的参数值。

示例

```
<form action="index.jsp">
<input name="name" type="text">
<input type="submit" value="提交">
</form>
<%
        String name = request.getParameter("name");
        out.println("用户姓名是：" + name);
%>
```

在 JSP 页面的文本框中输入用户姓名，然后单击“提交”按钮，该用户名将显示在 JSP 页面中。

8.7.16 getParameterValues 方法——获取指定参数的所有值

获得指定参数的所有值，参数由 name 指定。该方法通常用于获取复选按钮的取值。

方法声明

```
String[] getParameterValues(String name)
```

入口参数：name 是要获取的参数名称。

返回值：String 类型的字符串数组，该数组包含 name 参数指定的参数名称的所有参数值。

示例

```
<%@ page language="java" contentType="text/html; charset=GBk"
        pageEncoding="GBK"%>
        <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
TR/html4/loose.dtd">
        <head>
        <title>getParameterValues 方法示例</title>
        </head>
        <body>
        <form action="index.jsp" method="post">
        <input name="like" type="checkbox" value="体育">体育
```

```
<input name="like" type="checkbox" value="新闻">新闻<br>
<input name="like" type="checkbox" value="杂志">杂志
<input name="like" type="checkbox" value="经贸">经贸<br>
<input type="submit" value="提交"></form>
<%
        request.setCharacterEncoding("gbk");
        String[] likes = request.getParameterValues("like");
        if(likes==null)
                return;
        out.println("用户的爱好包括");
        for (int i = 0; i < likes.length; i++) {
                out.println("<br>"+likes[i]);
        }
%>
</body>
</html>
```

在 JSP 页面选择复选按钮之后，单击“提交”按钮，页面将显示用户选择的内容，如图 8.1 所示。

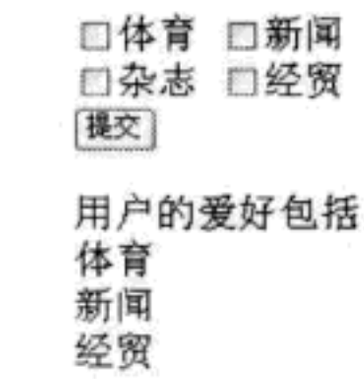


图 8.1　显示内容

8.7.17　getPathInfo 方法——获取客户请求 URL 的路径信息

返回客户请求 URL 的路径信息。

方法声明

```
String getPathInfo()
```

返回值：客户请求 URL 的路径信息，如果没有信息则返回 null。

示例

```
<%
        out.println(request.getPathInfo());
%>
```

8.7.18　getPathTranslated 方法——获取 Servlet 的附加路径信息

返回 URL 中在 Servlet 名称之后，在检索字符串之前的附加路径信息。如果在 URL 的 Servlet 路径后，没有附加路径信息，将返回空值。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
String getPathTranslated()
```

返回值：URL 的路径信息，如果没有信息则返回 null。

示例

```
<%
        out.println(request.getPathTranslated());
%>
```

8.7.19 getProtocol 方法——获取数据传送协议

该方法用于获取客户端向服务器端传送数据所依据的协议名称。

方法声明

```
String getProtocol()
```

返回值：协议名称。

示例

```
<%
        String p=request.getProtocol();
        out.println("使用的协议为："+p);
%>
```

该示例的运行结果将在 JSP 页面上显示“使用的协议为：HTTP/1.1”。

8.7.20 getQueryString 方法——获得查询字符串

该方法用于获得查询字符串，该字符串是由客户端以 get 方法向服务器端传送的。

方法声明

```
String getQueryString()
```

返回值：客户端以 get 方法提交的字符串。

示例

```
<a href="index.jsp?JavaCore">以 Get 方法发送的查询字符串</a><br>
<%
        String p=request.getQueryString();
        out.println("查询字符串为："+p);
%>
```

单击页面的超链接，该示例将在 JSP 页面输出“查询字符串为：JavaCore”。

8.7.21 getRemoteUser 方法——获取请求用户的登录信息

返回请求用户的登录信息，主要应用在 servlet 安全机制中检查用户是否已经登录。

方法声明

```
String getRemoteUser()
```

返回值：请求用户的登录信息，如果用户未通过认证则返回 null。

示例

```
<%
    out.println(request.getRemoteUser());
%>
```

8.7.22 getRequestURI 方法——获得客户端相对路径

该方法可以获得客户端访问网站的相对路径，即 URI 路径。

方法声明

```
String getRequestURI()
```

返回值：客户端访问网站的 URI 路径，返回类型为字符串。

示例

```
<%
    String p=request.getRequestURI();
    out.println("访问的 URI 路径为："+p);
%>
```

该示例将在 JSP 页面输出用户访问的 URI 地址：“访问的 URI 路径为：/example/index.jsp”。

8.7.23 getRequestURL 方法——获得客户端相对路径

该方法可以获得客户端访问网站的相对路径，即 URL 路径。

方法声明

```
StringBuffer getRequestURL()
```

返回值：客户端访问网站的 URL 路径，返回值类型为 StringBuffer。

示例

```
<%
    StringBuffer p=request.getRequestURL();
    out.println("访问的 URL 路径为："+p);
%>
```

该示例将在 JSP 页面输出用户访问的 URL 地址：“访问的 URL 路径为：http://localhost:8080/example/index.jsp”。

8.7.24 getRemoteAddr 方法——获取客户端的 IP 地址

该方法用于获取客户端的 IP 地址。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
String getRemoteAddr()
```

返回值：客户端 IP 地址的字符串表现形式。

示例

```
<%
        String p=request.getRemoteAddr();
        out.println("客户端的 IP 地址为："+p);
%>
```

该示例将在 JSP 页面输出：“客户端的 IP 地址为：127.0.0.1”。

8.7.25 getRemoteHost 方法——获取客户端的机器名称

该方法用于获取客户端的机器名称。

方法声明

```
String getRemoteHost()
```

返回值：客户端的主机名称。

示例

```
<%
        String p=request.getRemoteHost();
        out.println("客户端的主机名称为："+p);
%>
```

该示例将在 JSP 页面输出：“客户端的主机名称为：127.0.0.1”。

8.7.26 getRequestedSessionId 方法——获得客户端访问网站的 session ID 值

该方法可以获得客户端访问网站的 session ID 值。

方法声明

```
String getRequestedSessionId()
```

返回值：返回客户端的 session ID 值。

示例

```
<%
        out.println(request.getRequestedSessionId());
%>
```

8.7.27 getSession 方法——获取和请求相关的 session 对象

该方法用于获取和请求相关的 session 对象。

方法声明

```
HttpSession getSession(boolean create)
```

入口参数：create 参数是可选的，当有参数 create 且这个参数的值为 true 时，如果客户端还没有创建 session，那么将创建一个新的 session。

返回值：客户端的 session 对象。

示例

```
<%
    HttpSession ses=request.getSession(true);
    out.println("客户端 session 的 ID 编号为： "+ses.getId());
%>
```

该示例将在 JSP 页面输出 session 的 ID 编号。

8.7.28 getServerName 方法——获取服务器的名字

该方法用于获取服务器的名字。

方法声明

```
String getServerName()
```

返回值：服务器的名称。

示例

```
<%
    String serName=request.getServerName();
    out.println("服务器端的名字为： "+serName);
%>
```

该示例将在 JSP 页面输出“服务器端的名字为：localhost”。

8.7.29 getServerPath 方法——获取请求页面的文件路径

该方法用于获取客户端所请求页面的文件路径。

方法声明

```
String getServletPath()
```

返回值：客户端访问的页面路径，返回类型为 String 字符串。

示例

```
<%
    String path=request.getServletPath();
    out.println("访问的页面文件的路径为： "+path);
%>
```

该示例将在 JSP 页面输出“访问的页面文件的路径为：/index.jsp”。

8.7.30 getServerPort 方法——获取服务器的端口号

该方法用于获取服务器的端口号。

方法声明

```
int getServerPort()
```


PDF电子书基地 http://dayo1982.400gb.com

返回值：服务器的端口号，返回类型为 int 类型。

示例

```
<%
    int port = request.getServerPort();
    out.print("服务器的端口号为:"+port);
%>
```

该示例将在 JSP 页面输出“服务器的端口号为:8080”。

8.7.31 getUserPrincipal 方法——返回 Principal 对象

返回 java.security.Principal 对象，包括当前登录用户名称。

方法声明

```
Principal getUserPrincipal()
```

返回值：返回 java.security.Principal 对象，如果用户未能通过认证返回 null。

示例

```
<%
        Principal p = request.getUserPrincipal();
%>
```

8.7.32 isRequestedSessionIdFromCookie 方法——判断 session ID 是否从 Cookie 中获得

该方法返回 session ID 是否从 Cookie 中获得的布尔值。

方法声明

```
boolean isRequestedSessionIdFromCookie()
```

返回值：session ID 从 Cookie 中获得返回 true，否则返回 false。

示例

输出当前 session ID 是否从 Cookie 中获得。

```
<%
        out.println("from cookie" + request.isRequestedSessionIdFromCookie());
%>
```

8.7.33 isRequestedSessionIdFromURL 方法——判断 session ID 是否从 URL 中获得

该方法返回 session ID 是否从 URL 中获得的布尔值。

方法声明

```
boolean isRequestedSessionIdFromURL()
```

返回值：session ID 从 URL 中获得返回 true，否则返回 false。

示例

输出当前 session ID 是否从 URL 中获得。

```
<%
        out.println("from URL" + request.isRequestedSessionIdFromURL());
%>
```

8.7.34 isRequestedSessionIdValid 方法——判断返回 session 是否有效

该方法判断返回 session 是否有效。

方法声明

```
boolean isRequestedSessionIdValid()
```

返回值：返回 session 是否有效。

示例

输出当前 session 是否有效。

```
<%
        out.println("Session  是否有效" + request.isRequestedSessionIdValid());
%>
```

8.7.35 isUserInRole 方法——判断认证后的用户是否属于逻辑的 role 中的成员

判断认证后的用户是否属于逻辑 role 中的成员。

方法声明

```
boolean isUserInRole(String role)
```

入口参数：role 是一个字符串，是验证的角色名称。

返回值：布尔值。

示例

```
<%
        out.println("用户 lzw 是 role 中的成员吗？ : " + request.isUserInRole("lzw"));
%>
```

该示例将在 JSP 页面输出：“用户 lzw 是 role 中的成员吗？ : false”。

8.7.36 removeAttribute 方法——删除请求中的一个属性

该方法用于删除请求中的一个属性。

方法声明

```
void removeAttribute(String name)
```

入口参数：name 是要移除的属性的名称。

示例

```
<%
        request.removeAttribute("name");
        request.removeAttribute("sex");
```


PDF电子书基地 http://dayo1982.400gb.com

```
        request.removeAttribute("age");
        request.removeAttribute("city");
        request.removeAttribute("tel");
    %>
```

8.7.37 setAttribute 方法——设置 request 对象的属性

该方法用于设置 request 对象的属性。

方法声明

```
void setAttribute(String name, Object o)
```

入口参数

name 是要设置的属性名称。

o 该属性的属性值。

示例

```
<%
        String uName = "tom";
        String uSex = "man";
        String uAge = "26";
        String uCity = "长春";
        String uTel = "13999999";
        request.setAttribute("name", uNname);
        request.setAttribute("sex", uSex);
        request.setAttribute("age", uAge);
        request.setAttribute("city", uCity);
        request.setAttribute("tel", uTel);
%>
```

8.8 response 对象——响应对象

response 对象和 request 对象的性质相反，它所包含的是服务器向客户端做出的应答信息。实现过程中 response 被包装成 HttpServletResponse 接口的实现，它封装了 JSP 产生的响应，然后被发送到客户端以响应客户端的请求。response 对象包含以下几种内容：MIME 类型的文件，编码方式、保存 Cookie、链接到其他 URL。原理是它由容器生成，作为 service 方法的参数被传入 JSP，因为输出流是缓冲的，因此，可以设置 Http 状态码和 response 头。

8.8.1 addCookie 方法——保存客户端的 Cookie 信息

该方法用于添加 cookie 对象，用来保存客户端的用户信息。

方法声明

```
void addCookie(Cookie cookie)
```

入口参数：cookie 是要添加的 Cookie 对象。

示例

```
<%
    Cookie cookie=new Cookie("user","lzw");
    response.addCookie(cookie);
%>
```

8.8.2 addDateHeader 方法——加入对应名称的日期头信息

该方法用于加入对应名称的日期头信息，其中日期用的 long 类型的值表示，其值是从新纪元开始算起的毫秒数。

方法声明

```
void addDateHeader(String name, long date)
```

入口参数：name 是 header 的名称。

date 是要设置的 long 类型的数值。

示例

```
<%
    long date=System.currentTimeMillis();
    response.addDateHeader("Expires",date);
%>
```

8.8.3 addHeader 方法——添加 Http 文件头信息

该方法用于添加 Http 文件头信息，该 Header 将传送到客户端去，如果已经存在同名的 Header，则将会覆盖已有的 Header。

方法声明

```
void addHeader(String name, String value)
```

入口参数

name 是 header 的名称。

value 是 header 的值。

示例

```
<%=new Date() %>
<%
    response.addHeader("Refresh","1");
%>
```

8.8.4 addIntHeader 方法——加入对应名称的 int 属性

该方法用于加入对应名称的 int 属性。

方法声明

```
void addIntHeader(String name, int value)
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：name 是 header 的名称。

value 是整数值。

示例

```
<%
        response.addIntHeader("Expires",10);
%>
```

8.8.5 containsHeader 方法——判断 Http 文件头是否存在

该方法用于判断指定名字的 Http 文件头是否已经存在。

方法声明

```
boolean containsHeader(String name)
```

入口参数：name 是要判断的 header 的名称。

返回值：如果指定的 header 已经存在，返回 true，否则，返回 false。

示例

```
<%=new Date() %>
<%
        response.addHeader("Refresh","1");
        boolean b=response.containsHeader("Refresh");
        out.println("Refresh:"+b);
%>
```

8.8.6 getCharacterEncoding 方法——设置字符编码

该方法用于获得此时响应对象所采用的字符编码类型。

方法声明

```
String getCharacterEncoding()
```

返回值：响应对象采用的编码格式。

示例

```
<%
        String encode=response.getCharacterEncoding();
        out.println("响应客户端的编码格式为："+encode);
%>
```

8.8.7 getOutputStream 方法——获取 JSP 应答输出流

该方法用于获取 JSP 应答对象的输出流。

方法声明

```
ServletOutputStream getOutputStream() throws IOException
```

返回值：JSP 页面的输出流对象。

抛出异常：IOException。

示例

```
<%
        ServletOutputStream os= response.getOutputStream();
        os.write("<b>Java 编程词典</b>".getBytes());
        os.close();
%>
```

8.8.8 sendError 方法——发送错误信息

向客户端发送错误信息，例如：404 是指网页不存在或者请求的页面无效。

方法声明

```
void sendError(int sc) throws IOException
```

入口参数：sc 是要发送的错误代码。

抛出异常：IOException。

示例

```
<%
        response.sendError(404);
%>
```

该示例将在 JSP 页面输出 404 的错误页面，如图 8.2 所示。

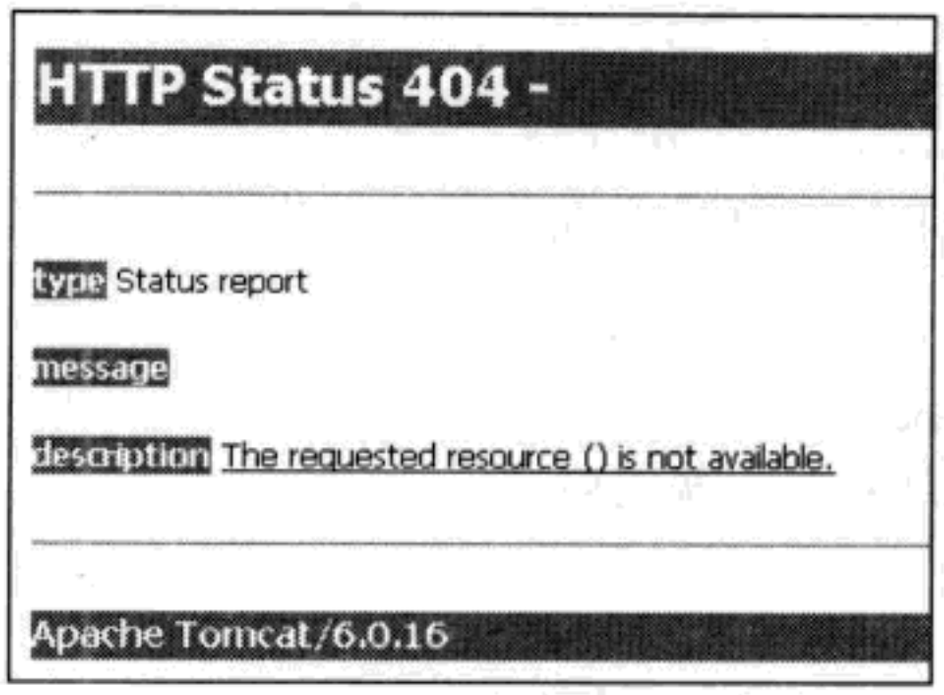


图 8.2 错误页面

8.8.9 sendRedirect 方法——响应重定向

该方法用于把响应重定向到另一个位置进行处理。

方法声明

```
void sendRedirect(String location) IOException
```

入口参数：location 是要重定向的地址。

抛出异常：IOException。

PDF电子书基地 http://dayo1982.400gb.com

示例

```
<%
    response.sendRedirect("http://www.mrbccd.com");
%>
```

8.8.10 setContentType 方法——设置 MIME 类型

该方法用于设置 JSP 响应对象的 MIME 类型。

方法声明

```
void setContentType(String type)
```

入口参数：type 是要设置的 MIME 类型。

示例

```
<%
    response.setContentType("video/x-msvideo");
%>
```

8.8.11 setDateHeader 方法——头名称及日期设置为响应头信息

该方法把指定的头名称以及日期设置为响应头信息，其中日期用的 long 类型的值表示，其值是从新纪元开始算起的毫秒数。

方法声明

```
void setDateHeader(String name, long date)
```

入口参数：name 是 header 的名称。

date 是要设置的 long 类型的数值。

示例

```
<%
    long date=System.currentTimeMillis();
    response.setDateHeader("Expires",date);
%>
```

8.8.12 setHeader 方法——设置头信息

此方法使用指定的头名称以及相应的值来设置头信息。如果头信息已经设置，则新的值会覆盖旧的值；如果头信息已经被发送出去，则此方法的设置将被忽略。

方法声明

```
void setHeader(String name,   value)
```

入口参数：name 是要设置的 header 的名称。

value 是该名称对应的值。

示例

```
<%
    out.println(new Date());
    response.setHeader("Refresh","1");
%>
```

8.8.13 setStatus 方法——设置响应的状态码

该方法用于设置响应的状态编码。

方法声明

```
void setStatus(int sc)
```

入口参数：sc 是状态代码。

示例

```
<%
response.setStatus(200);
%>
```

8.9 session 对象——会话对象

session 是用于保存客户信息而分配给客户的对象，HTTP 协议不能保存客户端请求信息的历史记录，因此为了弥补这一缺点，生成一个 session 对象，这样服务器和客户端之间的连接就会一直保持下去，但是在一定时间内，如果客户端不向服务器发出应答请求，系统默认在 30 分钟内，session 对象就会自动消失。但是在编写程序时，可以修改这个时间限定值，使 session 对象在特定时间内保存信息。

保存的信息可以是与客户端有关的，也可以是一般信息，程序员要决定相应信息的内容。在许多服务器上，如果浏览器支持 cookies 的话，就直接使用 cookies，但是如果不支持或废除了 cookies 的话，就自动转化为 URL（重写 URL，这个 URL 包含客户端的信息），session 自动为每个流程提供了方便的存储信息的方法。JSP 的 session 对象通过 JSP 容器变换为 javax.servlet.http.HttpSession 对象。所以，session 对象和 Servlet 的 javax.servlet.http.HttpSession 对象起到相同的作用。

8.9.1 getAttribute 方法——获取属性值

该方法用于获取与指定名字相联系的属性值。

方法声明

```
Object getAttribute(String name)
```

入口参数：name 是要获取的属性名称。

返回值：name 指定的属性值。

PDF电子书基地 http://dayo1982.400gb.com



示例

```
<%
session.getAttribute("user");
%>
```

8.9.2 getAttributeNames 方法——获取属性名称

该方法用于获取 session 对象中存储的每一个属性对象的名称，其结果为一个枚举类的对象。

方法声明

```
Enumeration getAttributeNames()
```

返回值：保存 session 对象的所有属性名称的枚举对象。

示例

```
<%
        session.setAttribute("user","lzw");
        session.setAttribute("age","27");
        session.setAttribute("addr","Jilin");
        Enumeration names = session.getAttributeNames();
        while (names.hasMoreElements()) {
                out.println(names.nextElement() + "<br>");
        }
%>
```

8.9.3 getCreationTime 方法——获取 session 对象创建时间

该方法用于获取 session 对象被创建的时间，最小单位为千分之一秒。

方法声明

```
long getCreationTime()
```

返回值：创建 session 对象的时间的 long 数字。

示例

```
<%
        session.setAttribute("user", "lzw");
        long timenum = session.getCreationTime();
        out.println("session 的创建时间是：" +new Date(timenum));
%>
```

8.9.4 getId 方法——获取惟一的 ID 标识

该方法用于获取惟一的 ID 标识，每个 session 的 ID 是不同的。

方法声明

```
String getId()
```

返回值：session 对象的唯一 ID。

示例

```
<%
    String ID=session.getId();
    out.println("这个 session 的 ID 是：" +ID);
%>
```

8.9.5 getLastAccessedTime 方法——获取最后发送请求的时间

该方法用于获取和当前 session 对象相关的客户端最后发送请求的时间，最小单位为千分之一秒。

方法声明

```
long getLastAccessedTime()
```

返回值：session 对象最后访问时间的 long 值。

示例

```
<%
    long timeNum=session.getLastAccessedTime();
    Date date=new Date(timeNum);
    out.println("这个 session 的最后访问时间是：" +date);
%>
```

8.9.6 getMaxInactiveInterval 方法——获取 session 有效时间

该方法用于获取 session 的有效时间，负值表示 session 永远不会超时，它的值为该 session 对象的生存时间。

方法声明

```
int getMaxInactiveInterval()
```

返回值：session 的生存时间，单位是 s。

示例

```
<%
    long timeNum=session.getMaxInactiveInterval();
    out.println("这个 session 的有效时间是：" +timeNum);
%>
```

8.9.7 getServletContext 方法——获取 ServletContext 对象

返回 session 所属的 ServletContext 对象。

方法声明

```
ServletContext getServletContext()
```

返回值：ServletContext 对象。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

示例

```
<%
ServletContext sc = session.getServletContext();
%>
```

8.9.8 invalidate 方法——销毁 session 对象

销毁这个 session 对象，使得和它绑定的对象都失效。

方法声明

```
void invalidate()
```

示例

```
<%
session. invalidate();
%>
```

8.9.9 isNew 方法——判断当前用户是否为新用户

该方法是判断当前用户是否为新用户，可以判断用户是否刷新了当前页面。它的返回值为 boolean 型。

方法声明

```
boolean isNew()
```

返回值：如果当前用户是否为新用户返回 true，否则返回 false。

示例

```
<%
out.print(session.isNew());
%>
```

8.9.10 setAttibute 方法——设置属性值

该方法用于设置指定名称的属性值，并将其存储在 session 对象中。

方法声明

```
void setAttribute(String name, Object value)
```

入口参数：name 是要设置的属性名称。

value 是对应属性名称的属性值。

示例

```
<%
        session.setAttribute("information","向 session 中保存数据");
        response.sendRedirect("forward.jsp");
%>
```

8.9.11 setMaxInactiveInterval 方法——设置 session 有效时间

该方法用于设置保存 session 数据的时间。

方法声明

```
void setMaxInactiveInterval(int interval)
```

入口参数：interval 是保存的秒数。

示例

```
<%
    session.setMaxInactiveInterval(30);
%>
```

8.9.12 removeAttribute 方法——移除指定属性

该方法用于删除指定的 session 属性。

方法声明

```
void removeAttribute(String name)
```

入口参数：name 是要移除的属性的名称。

示例

```
<%
    session.removeAttribute("user");
%>
```


PDF电子书基地 http://dayo1982.400gb.com

第9章　JSTL 核心标签库

9.1　表达式标签

JSTL 的核心标签库中的表达式标签主要用来输出表达式的值、设置或移除变量以及处理异常。

9.1.1　<c:out>标签——输出标签

<c:out>标签用于将计算的结果输出到 JSP 页面中，该标签可以替代<%=%>。

说明

<%=%>是 JSP 使用程序代码输出表达式或变量结果的常用方式，但是在 JSP 网页中嵌入 Java 程序代码已经不再推荐使用，取而代之的是使用 JSTL 标签来完成优雅的 JSP 页面编码，而且这样做还可以是页面代码整洁，提高了代码的可读性。

<c:out>标签有两种语法格式，这两种语法格式的输出结果完全相同。

语法 1

```
<c:out value="value" [escapeXml="true|false"] [default="defaultValue"]/>
```

语法 2

```
<c:out value="value" [escapeXml="true|false"]>
    defalultValue
</c:out>
```

<c:out>标签的各属性的详细介绍如下表所示。

<c:out>标签属性表

属　性	类　型	描　述	引用 EL
value	Object	将要输出的变量或表达式	可以
escapeXml	boolean	转换特殊字符，默认值为 true，例如“<”转换为“<”	不可以
default	Object	如果 value 属性值等于 NULL，则显示 default 属性定义的默认值	不可以

示例

测试<c:out>标签的 escapeXml 属性及通过两种语法格式设置 default 属性时的显示结果，具体代码如下：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
escapeXml 属性值为 false 时：<c:out value="<hr>" escapeXml="false"/>
```

```
escapeXml 属性值为 true 时：<c:out value="<hr>"/>
第一种语法格式：<c:out value="${name}" default="name 的值为空"/>
<br>
第二种语法格式：<c:out value="${name}">
 name 的值为空
</c:out>
```

运行结果如图 9.1 所示。

```
escapeXml属性值为false时：
escapeXml属性值为true时：<hr> 第一种语法格式：name的值为空
第二种语法格式：name的值为空
```

图 9.1　运行结果

9.1.2　<c:set>标签——设置标签

<c:set>标签用于定义和存储变量，它可以定义变量是在 JSP 会话范围内还是 JavaBean 的属性中，可以使用该标签在页面中定义变量，而不用在 JSP 页面中嵌入打乱 HTML 排版的 Java 代码。<:set>标签有 4 种语法格式。

语法 1

该语法格式在 scope 指定的范围内将变量值存储到变量中。

```
<c:set value="value" var="name" [scope="page|request|session|application"]/>
```

语法 2

该语法格式在 scope 指定的范围内将标签主体存储到变量中。

```
<c:set var="name" [scope="page|request|session|application"]>
标签体
</c:set>
```

语法 3

该语法格式将变量值存储在 target 属性指定的目标对象的 propName 属性中。

```
<c:set value="value" target="object" property="propName"/>
```

语法 4

该语法格式将标签主体存储到 target 属性指定的目标对象的 propName 属性中。

```
<c:set target="object" property="propName">
标签体
</c:set>
```

<c:set>标签的各属性的详细介绍如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

<c:set>标签属性表

属　性	类　型	描　述	引用 EL
value	Object	将要存储的变量值	可以
var	String	存储变量值的变量名称	不可以
target	Object	存储变量值或者标签主体的目标对象，可以是 JavaBean 或 Map 集合对象	可以
property	String	指定目标对象存储数据的属性名	可以
scope	String	指定变量存在于 JSP 的范围，默认值是 page	不可以

示例

应用<c:set>标签定义不同范围内的变量。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="name" value="编程体验网" scope="page"/>
<c:set var="hostpage" value="www.bcty365.com" scope="session"/>
```

9.1.3 <c:remove>标签——移除标签

<c:remove>标签可以从指定的 JSP 范围中移除指定的变量。

语法

```
<c:remove var="name" [scope="page|request|session|application"]/>
```

属性：var 用于指定存储变量值的变量名称。

scope 用于指定变量存在于 JSP 的范围，可选值有 page、request、session、application。默认值是 page。

示例

应用<c:set>标签定义一个 page 范围内的变量，然后输出该变量，再应用<c:remove>标签移除该变量，最后再输出该变量。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="name" value="编程体验网" scope="page"/>
移除前输出的变量 name 为：<c:out value="${name}"></c:out>
<c:remove var="name"/>
<br>
移除后输出的变量 name 为：<c:out value="${name}" default="变量 name 为空"></c:out>
```

运行结果如图 9.2 所示。

```
移除前输出的变量name为：编程体验网
移除后输出的变量name为：变量name为空
```

图 9.2　运行结果

9.1.4 <c:catch>标签——捕获异常标签

<c:catch>标签是 JSTL 中处理程序异常的标签，它还能够将异常信息保存在

变量中。

语法

```
<c:catch [var="name"]>
……存在异常的代码
</c:catch>
```

属性：var 为可以指定存储异常信息的变量。这是一个可选项，如果不需要保存异常信息，可以省略该属性。

示例

在 JSP 页面中经常会出现各种异常信息，这可能是程序中的错误，也可能是输入数据的错误，如果在 JSP 页面中可以处理这些错误信息，就可以避免它们破坏页面的内容和美观。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:catch var="error">
<%
String str="wgh";
Integer number=new Integer(str);
%>
</c:catch>
<c:out value="发生异常${error }"></c:out>
```

运行上面的代码，将在页面中输出以下字符串：

```
发生异常 java.lang.NumberFormatException: For input string: "wgh"
```

9.2 流程控制标签

流程控制在程序中会根据不同的条件去执行不同的代码来产生不同的运行结果，使用流程控制可以处理程序中的任何可能发生的事情。在 JSTL 中包含<c:if>标签、<c:choose>标签、<c:when>标签和<c:otherwise>标签 4 种流程控制标签。

9.2.1 <c:if>标签——条件判断标签

这个标签可以根据不同的条件去处理不同的业务，也就是执行不同的程序代码。它和 Java 基础中 if 语句的功能一样。<c:if>标签有两种语法格式。

语法 1

该语法格式会判断条件表达式，并将条件的判断结果保存在 var 属性指定的变量中，而这个变量存在于 scope 属性所指定范围中。

```
<c:if test="condition" var="name" [scope=page|request|session|application]/>
```

语法 2

该语法格式不但可以将 test 属性的判断结果保存在指定范围的变量中，还可以根据条件的判断结果去执行标签主体。标签主体可以是 JSP 页面能够使用的任

PDF电子书基地 http://dayo1982.400gb.com

何元素，例如 HTML 标记、Java 代码或者嵌入其他 JSP 标签。

```
<c:if test="condition" var="name" [scope=page|request|session|application]>
    标签体
</c:if>
```

<c:if>标签的各属性的详细介绍如下表所示。

<c:if>标签属性表

属　性	类　型	描　述	引用 EL
test	boolean	条件表达式，这是<c:if>标签必须定义的属性	可以
var	String	指定变量名，这个属性会指定 test 属性的判断结果将存放在那个变量中，如果该变量不存在就创建它	不可以
scope	String	存储范围，该属性用于指定 var 属性所指定的变量的存在范围	不可以

示例

应用<c:if>标签判断用户名是否为空，如果为空则显示一个用于输入用户名的文本框及“提交”按钮。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
语法一：输出用户名是否为 null<br>
    <c:if test="${param.user==null}" var="rtn" scope="page"/>
    <c:out value="${rtn}"/>
<br>语法二：如果用户名为空，则输出一个用于输入用户名的文本框及“提交”按钮<br>
<c:if test="${param.user==null}">
    <form action="" method="post">
        请输入用户名：<input type="text" name="user">
        <input type="submit" value="提交">
    </form>
</c:if>
```

运行本程序，当用户名为空时，将显示如图 9.3 所示的运行结果。

输入用户名后，单击“提交”按钮，将显示如图 9.4 所示的运行结果。

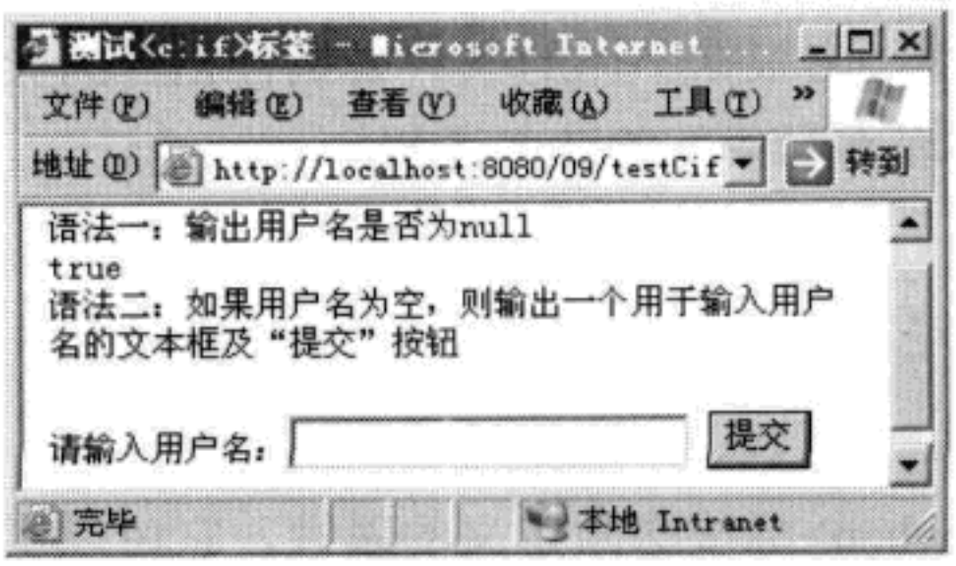


图 9.3　用户名为空的运行结果

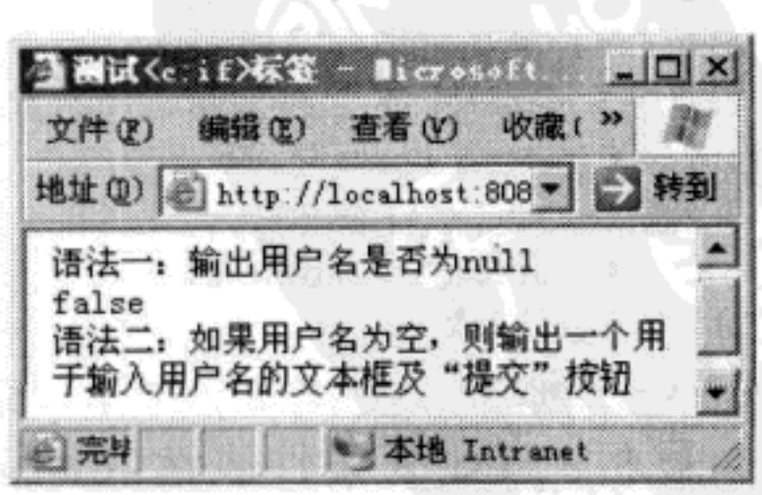


图 9.4　用户名不为空的运行结果

9.2.2 <c:choose>标签——选择标签

<c:choose>标签可以根据不同的条件去完成指定的业务逻辑，如果没有符合的条件会执行默认条件的业务逻辑。<c:choose>标签只能作为<c:when>和<c:otherwise>标签的父标签，可以在它之内嵌套这两个标签完成条件选择逻辑。

语法

```
<c:choose>
    <c:when>
        业务逻辑
    </c:when>
    …        <!--多个<c:when>标签-->
    <c:otherwise>
        业务逻辑
    </c:otherwise>
</c:choose>
```

说明

<c:choose>标签中可以包含多个<c:when>标签来处理不同条件的业务逻辑，但是只能有一个<c:otherwise>标签来处理默认条件的业务逻辑。

示例

应用<c:choose>标签提示当前日期是休息日还是工作日。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="day">
    <%=new java.util.Date().getDay()%>
</c:set>
<c:choose>
    <c:when test="${day==6}">今天是星期六，可以去逛街了！</c:when>
    <c:when test="${day==0}" >今天是星期日，可以在家休息了！</c:when>
    <c:otherwise>今天是工作，努力工作吧！</c:otherwise>
</c:choose>
```

当周星期一运行上面的代码时，将显示以下内容：

```
今天是工作日，努力工作吧！
```

9.2.3 <c:when>标签——条件标签

这是包含在<c:choose>标签的子标签，它根据不同的条件去执行相应的业务逻辑，可以存在多个<c:when>标签来处理不同条件的业务逻辑。

语法

```
<c:when test="condition">
……标签体
</c:when>
```

属性：test 为条件表达式，这是<c:when>标签必须定义的属性，它可以引用 EL 表达式。

PDF电子书基地 http://dayo1982.400gb.com

注意

<c:when>标签必须出现在<c:otherwise>标签之前。

示例

应用<c:choose>标签和<c:when>标签实现分时问候。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="hours">
    <%=new java.util.Date().getHours()%>
</c:set>
<c:choose>
    <c:when test="${hours>1 && hours<6}">早上好！ </c:when>
    <c:when test="${hours>6 && hours<11}" >上午好！ </c:when>
    <c:when test="${hours>11 && hours<17}">下午好！ </c:when>
    <c:when test="${hours>17 && hours<24}">晚上好！ </c:when>
</c:choose>
 现在时间是：${hours}时
```

当 13 时 36 分时，运行上面的代码将显示以下内容：

```
下午好！  现在时间是：13 时
```

9.2.4 <c:otherwise>标签——指定默认处理逻辑标签

<c:otherwise>标签也是一个包含在<c:choose>标签的子标签，用于定义<c:choose>标签中的默认条件处理逻辑，如果没有任何一个结果满足<c:when>标签指定的条件，将会执行这个标签主体中定义的逻辑代码。在<c:choose>标签范围内只能存在一个该标签的定义。

语法

```
<c:otherwise>
标签体
</c:otherwise>
```

属性：test 为条件表达式，这是<c:when>标签必须定义的属性，它可以引用 EL 表达式。

注意

<c:otherwise>标签必须定义在所有<c:when>标签的后面，也就是说它是<c:choose>标签的最后一个子标签。

示例

应用<c:choose>标签、<c:when>标签和<c:otherwise>标签显示随机抽奖结果。

```
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%Random rnd=new Random();%>
<c:set var="luck">
    <%=rnd.nextInt(10)%>
```

```
    </c:set>
    <c:choose>
        <c:when test="${luck==6}">恭喜你，中了一等奖！</c:when>
        <c:when test="${day==7}" >恭喜你，中了二等将！</c:when>
        <c:otherwise>谢谢您的参与！</c:otherwise>
    </c:choose>
```

运行上面的代码，当随机生成数字 6 时，将显示以下内容：

```
恭喜你，中了一等奖！
```

9.3 循环标签

循环是程序算法中的重要环节，有很多著名的算法都需要在循环中完成，例如递归算法、查询算法和几乎所有的排序算法都需要在循环中完成。JSTL 标签库中包含<c:forEach>和<c:forTokens>两个循环标签。

9.3.1 <c:forEach>标签——对对象进行迭代输出标签

<c:forEach>标签可以根据循环条件，遍历数组和集合类中的所有或部分数据。例如，在使用 Hibernate 技术访问数据库时，返回的都是数组、java.util.List 和 java.util.Map 对象，它们封装着从数据库中查询出的数据，这些数据是 JSP 页面需要的。如果在 JSP 页面中使用 Java 代码来循环遍历所有数据，会使页面非常混乱，不易分析和维护。使用 JSTL 的<c:forEach>标签循环显示这些数据可以解决 JSP 页面混乱的问题，同时也提高了代码的可维护性。

语法

```
    <c:forEach items="data" var="name" begin="start" end="finish" step="step" varStatus="
statusName">
        ……标签体
    </c:forEach>
```

<c:forEach>标签中的属性都是可选项，可以根据需要使用相应的属性。<c:forEach>标签的各属性的详细介绍如下表所示。

<c:forEach>标签属性表

属　性	类　型	描　述	引用 EL
items	数组、集合类、字符串和枚举类型	被循环遍历的对象，多用于数组与集合类	可以
var	String	循环体的变量，用于存储 items 指定的对象的成员	不可以
begin	int	循环的起始位置	可以
end	int	循环的终止位置	可以
step	int	循环的步长	可以
varStatus	String	循环的状态变量	不可以

PDF电子书基地 http://dayo1982.400gb.com

在上面的属性中，varStatus 属性又包括如下表所示的 4 个当前循环体的状态属性。

varStatus 属性的 4 个状态属性

属　性	类　型	说　明
index	int	当前循环的索引值
count	int	循环总数
first	boolean	是否为第一次循环
last	boolean	是否为最后一次循环

示例

应用<c:forEach>标签遍历一个 java.util.List 集合对象，并将其显示在 JSP 页面中，具体代码如下：

```
<%@ page import="java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%
    List<String> list = new ArrayList<String>();
    list.add("表达式标签");
    list.add("流程控制标签");
    list.add("循环标签");
    list.add("URL 操作");
    request.setAttribute("data", list);
%>
List 集合类中包含了 JSTL 核心标签库的 4 种标签名。<br>
利用&lt;c:forEach&gt;标签遍历其结果如下：
<br>
<c:forEach items="${data}" var="tag" varStatus="id">
            ${id.count } ${tag }<br>
</c:forEach>
```

运行结果如图 9.5 所示。

List集合类中包含了JSTL核心标签库的4种标签名。
利用<c:forEach>标签遍历其结果如下：
1 表达式标签
2 流程控制标签
3 循环标签
4 URL操作

图 9.5　运行结果

9.3.2　<c:forTokens>标签——对字符串进行迭代输出标签

<c:forTokens>标签用于在JSP中遍历一个字符串中所有由定义符号所分隔的成员，当条件成立时，循环执行<c:forTokens>标签体中的代码段。

语法

```
<c:forTokens items="String" delims="char" [var="name"] [begin="start"] [end="end"] [step="len"] [varStatus="statusName"]>
……标签体
</c:forTokens>
```

<c:forTokens >标签的各属性的详细介绍如下表所示。

<c:forTokens >标签属性表

属　性	类　型	描　述	引用 EL
items	String	被循环遍历的对象，多用于数组与集合类	可以
delims	String	字符串的分割字符	不可以
var	String	变量名称	不可以
begin	int	循环的起始位置	可以
end	int	循环的终止位置	可以
step	int	循环的步长	可以
varStatus	String	循环的状态变量	不可以

示例

下面的实例演示了如何分割字符串，被分割的字符串有多种分割符。将字符串分割后会添加“
”标记换行显示。

```
<%@ page import="java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%
    String str = "软件涵盖技术、函数、控件、实例、项目、方案、界面等所有开发内容，以及所有实例程序、实用工具等内容，是程序开发人员高效编程必备的软件。";
    request.setAttribute("str", str);
%>
<c:forTokens items="${str }" delims="，、。" var="item">
        ${item}<br>
</c:forTokens>
```

运行结果如图 9.6 所示。

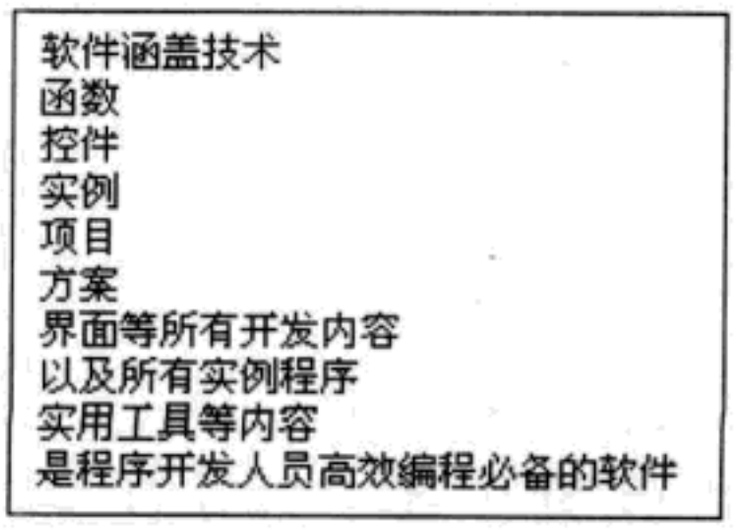


图 9.6　运行结果

PDF电子书基地 http://dayo1982.400gb.com

9.4 URL 操作标签

URL 操作标签是指与页面导航、重定向、资源获得、参数传递相关的标签。

9.4.1 <c:import>标签——文件导入标签

<c:import>标签可以导入站内或其他网站的静态和动态文件到 JSP 页面中，例如，使用<c:import>标签导入其他网站的天气信息到自己的 JSP 页面中。与此相比，<jsp:include>只能导入站内资源，<c:import>的灵活性要高很多。

语法 1

```
<c:import url="url" [context="context"] [var="name"]
 [scope="page|request|session|application"] [charEncoding="encoding"]
     ……标签体
</c:import>
```

语法 2

```
<c:import url="url" varReader="name" [context="context"] [charEncoding="encoding"]
```

这两种语法格式中涉及到的属性说明如下表所示。

<c: import >标签属性表

属　性	类　型	描　述	引用 EL
url	String	被导入的文件资源的 URL 路径	可以
context	String	上下文路径，用于访问同一个服务器的其他 Web 工程，其值必须以“/”开头，如果指定了该属性，那么 url 属性值也必须以“/”开头	可以
var	String	变量名称，将获取的资源存储在变量中	不可以
scope	String	变量的存在范围	不可以
varReader	String	以 Reader 类型存储被包含文件内容	不可以
charEncoding	String	被导入文件的编码格式	可以

示例

应用<c:import>标签导入编程词典简介的文本资源文件，然后插入到页面的文本区域中。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
编程词典简介<br>
<textarea name="artcle" cols="75" rows="14" class="textarea">
    <c:import url="agreement.txt" charEncoding="gbk" />
</textarea>
```

运行结果如图 9.7 所示。

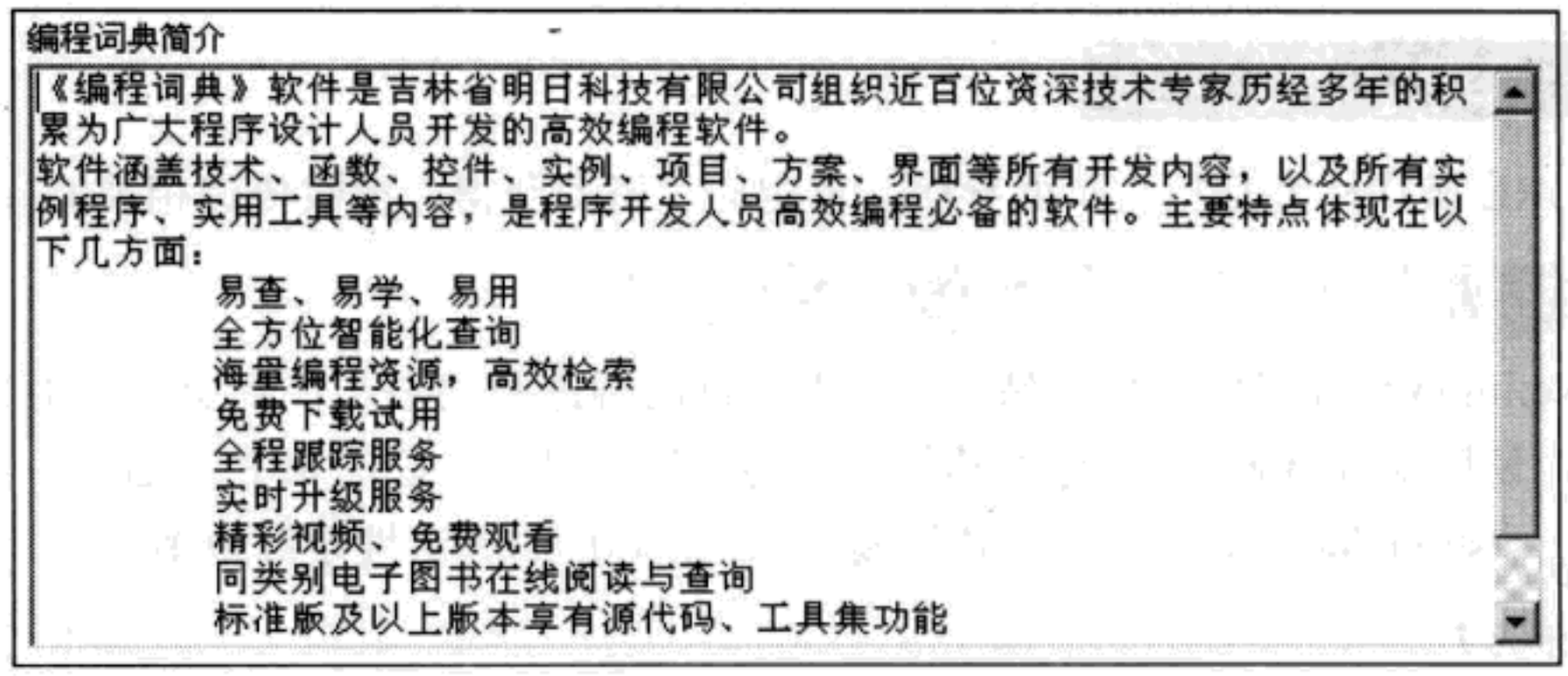


图 9.7 运行结果

9.4.2 <c:redirect>标签——重定向标签

<c:redirect>标签可以将客户端发出的 request 请求重定向到其他 URL 服务端，由其他程序处理客户的请求。而在这期间可以对 request 请求中的属性进行修改或添加，然后把所有属性传递到目标路径。

语法 1

```
<c:redirect url="url" [context="/context"]/>
```

该语法格式没有标签主体，并且不添加传递到目标路径的参数信息。

语法 2

```
<c:redirect url="url" [context="/context"]>
    ……    <c:param>
</c:redirect>
```

该语法格式将客户请求重定向到目标路径，并且在标签主体中使用<c:param>标签传递其他参数信息。

这两种语法格式中涉及到的属性说明如下表所示。

<c:redirect >标签属性表

属　性	类　型	描　述	引用 EL
url	String	重定向到该属性指定的目标路径，它是标签必须指定的属性	可以
context	String	url 属性的相对路径	可以

示例

应用<c:redirect >标签将页面重定向到 main.jsp 页面。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:redirect url="main.jsp"/>
```

运行上面的代码，将页面重定向到 main.jsp 页面，显示以下文字：

PDF电子书基地 http://dayo1982.400gb.com

这是通过<c:redirect>标签重定向的页面！

9.4.3 <c:url>标签——生成 URL 地址标签

<c:url>标签用于生成一个 URL 路径的字符串，这个生成的字符串可以赋予 HTML 的<a>标记实现 URL 的连接，或者用这个生成的 URL 字符串实现网页转发与重定向等。在使用该标签生成 URL 时还可以搭配<c:param>标签动态添加 URL 的参数信息。

语法 1

```
<c:urlvalue="url"[var="name"][scope="page|request|session|application"] [context="context"]/>
```

该语法将输出产生的 URL 字符串信息，如果指定了 var 和 scope 属性，相应的 URL 信息就不再输出，而是存储在变量中以备后用。

语法 2

```
<c:url value="url" var="name"] [scope="page|request|session|application"] [context="context"]>
    <c:param>
</c:url>
```

该语法不仅实现了语法格式 1 的功能，而且还可以搭配<c:param>标签完成带参数的复杂 URL 信息。

这两种语法格式中涉及的属性说明如下表所示。

<c:url >标签属性表

属　性	类　型	描　述	引用 EL
url	String	生成的 URL 路径信息	可以
context	String	上下文路径，用于访问同一个服务器的其他 Web 工程，其值必须以“/”开头，如果指定了该属性，那么 url 属性值也必须以“/”开头	可以
var	String	变量名称，将获取的资源存储在变量中	不可以
scope	String	变量的存在范围	不可以
context	String	url 属性的相对路径	可以

示例

将 Tomcat 的 URL 路径“http://localhost:8080”定制到标签的 url 变量中，这个变量存在于 session 范围内。另外还结合了<c:param>标签动态生成 3 个 URL 的参数信息，这 3 个参数信息分别是当前时间的时、分、秒。

```
<%@ page import="java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
    <c:set var="time" value="<%=new Date()%>"/>
    <c:url value="http://localhost:8080" var="url" scope="session">
        <c:param name="Hours" value="${time.hours}"/>
        <c:param name="Minutes" value="${time.minutes}"/>
        <c:param name="Seconds" value="${time.seconds}"/>
```

```
        </c:url>
        <a href=${url }>用产生的 URL 路径作为超链接的参数</a>
    %>
```

运行结果如图 9.8 所示。

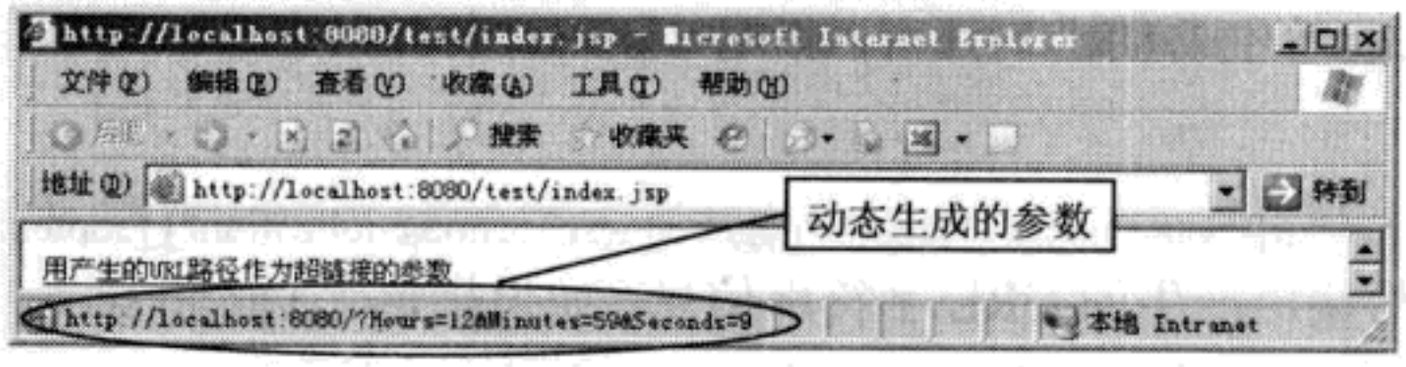


图 9.8 运行结果

9.4.4 <c:param>标签——参数传递标签

<c:param>标签只用于为其他标签提供参数信息，它与<c:import>标签、<c:redirect>标签和<c:url>标签组合可以实现动态定制参数，从而使标签可以完成更复杂的程序应用。

语法

```
<c:param name="paramName" value="paramValue"/>
```

<c:param >标签的各属性的详细介绍如下表所示。

<c:param >标签属性表

属　性	类　型	描　述	引用 EL
name	String	参数名称	可以
value	String	参数值	可以

示例

应用<c:redirect>标签及<c:param>实现将页面重定向到 main.jsp 页面，并传递两个参数。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:redirect url="main.jsp">
    <c:param name="user" value="wgh"/>
    <c:param name="pwd" value="111"/>
</c:redirect>
```

main.jsp 页面的代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
用户名为：${param.user}
<br>
密码为：${param.pwd}
```


PDF电子书基地 http://dayo1982.400gb.com

运行上面的示例，将页面重定向到 main.jsp 页面，并显示传递的参数，如图 9.9 所示。

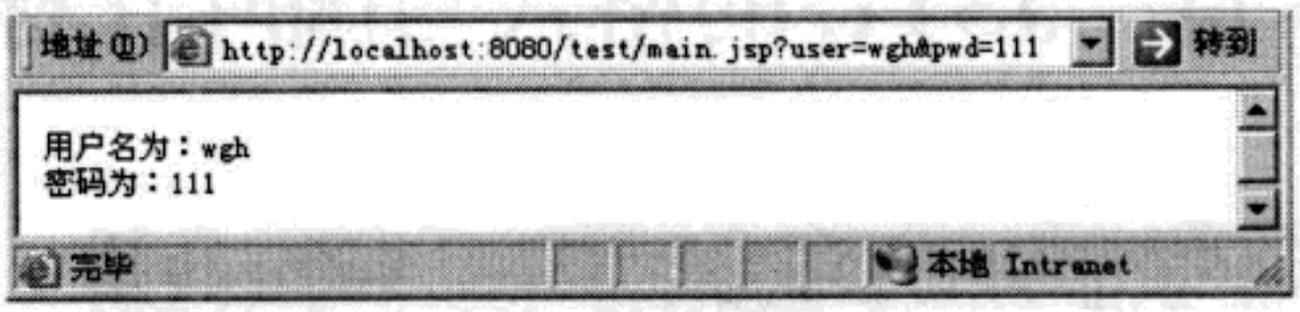


图 9.9　运行结果

第 10 章　JSTL 的格式与国际化标签库

10.1　<fmt:formatNumber>标签——格式化数字

<fmt:formatNumber>标签用于设置数字在不同国家区域的显示格式，例如，数字 1 000 000.9，在中国将使用 1,000,000.9 的格式，而在法国则会使用 1 000 000.9 的数字格式。

语法 1

```
<fmt:formatNumber value="numericValue"
[type="{number|currency|percent}"]
[pattern="customPattern"]
[currencyCode="currencyCode"]
[currencySymbol="currencySymbol"]
[groupingUsed="{true|false}"]
[maxIntegerDigits="maxIntegerDigits"]
[minIntegerDigits="minIntegerDigits"]
[maxFractionDigits="maxFractionDigits"]
[minFractionDigits="minFractionDigits"]
[var="varName"]
[scope="{page|request|session|application}"]/>
```

语法 2

```
<fmt:formatNumber [type="{number|currency|percent}"]
[pattern="customPattern"]
[currencyCode="currencyCode"]
[currencySymbol="currencySymbol"]
[groupingUsed="{true|false}"]
[maxIntegerDigits="maxIntegerDigits"]
[minIntegerDigits="minIntegerDigits"]
[maxFractionDigits="maxFractionDigits"]
[minFractionDigits="minFractionDigits"]
[var="varName"]
[scope="{page|request|session|application}"]>
numeric value to be formatted
</fmt:formatNumber>
```

<fmt:formatNumber>标签的各属性的详细介绍如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

<fmt:formatNumber>标签属性表

属　性	类　型	描　述	引用 EL
value	String，Number	被格式化的数字	可以
type	String	数字格式的表现类型，可以选择货币、百分比和数字类型。详细类型如下表所示	可以
pattern	String	模式字符串	可以
currencyCode	String	货币单位代码	可以
currencySymbol	String	货币符号	可以
groupingUsed	boolean	是否对格式化后数字的整数部分分组，例如，123,456,789.001	可以
maxIntegerDigits	int	数字格式化后的最大整数位数	可以
minIntegerDigits	int	数字格式化后的最小整数位数	可以
maxFractionDigits	int	数字格式化后的最大小数位数	可以
minFractionDigits	int	数字格式化后的最小小数位数	可以
var	String	存储格式化后的数字格式	不可以
scope	String	变量的作用范围	不可以

type 属性的类型表

类　型	说　明	示　例
number	数字格式	6.7
currency	货币格式	￥6.70
percent	百分比格式	670%

示例

应用<fmt:formatNumber>标签将数字 7.17 分别以数字、货币和百分比格式输出。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:formatNumber value="7.17" type="number"/>          <!-- 显示 7.17 -->
<br>
<fmt:formatNumber value="7.17" type="percent"/>         <!-- 显示 7.17 -->
<br>
<fmt:formatNumber value="7.17" type="currency"/>        <!-- 显示￥7.17 -->
```

示例

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- 显示 314,159.26 -->
<fmt:formatNumber value="314 159.26" groupingUsed="true" />
<!-- 显示 0457.17 -->
<fmt:formatNumber value="70 457.17" maxIntegerDigits="4" />
<!-- 显示 0,007.17 -->
<fmt:formatNumber value="7.17" minIntegerDigits="4" groupingUsed="true" />
<!-- 显示$7.17 -->
<fmt:formatNumber value="7.17"    currencySymbol="$" type="currency" />
<!-- 显示 3.14 -->
<fmt:formatNumber value="3.141 59" maxFractionDigits="2"/>
<!-- 显示 3.15 -->
<fmt:formatNumber value="3.145 59" maxFractionDigits="2"/>
```

示例

显示为科学计数法。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!-- 显示 3.142E-1 -->
<fmt:formatNumber value="0.314 159 265 358 97" pattern="0.###E0"/>
<!-- 显示 31.416E-2 -->
<fmt:formatNumber value="0.314 159 265 358 97" pattern="00.###E0"/>
<!-- 显示 3.142E13 -->
<fmt:formatNumber value="31 415 926 535 897" pattern="0.###E0"/>
```

10.2 <fmt:parseNumber>标签——转换为数字

<fmt:parseNumber>标签可以把字符串类型的数字解析成数字类型的数值，使其可以组合算术运算形成其他数值结果。

语法 1

```
<fmt:parseNumber value="numericValue"
[type="{number|currency|percent}"]
[pattern="customPattern"]
[parseLocale="parseLocale"]
[integerOnly="{true|false}"]
[var="varName"]
[scope="{page|request|session|application}"]/>
```

语法 2

```
<fmt:parseNumber [type="{number|currency|percent}"]
[pattern="customPattern"]
[parseLocale="parseLocale"]
[integerOnly="{true|false}"]
[var="varName"]
[scope="{page|request|session|application}"]>
```


PDF电子书基地 http://dayo1982.400gb.com

```
numeric value to be parsed
</fmt:parseNumber>
```

<fmt:parseNumber>标签的各属性的详细介绍如下表所示。

<fmt:parseNumber>标签属性表

属　性	类　型	描　述	引用 EL
value	String，Number	被解析的字符串	可以
type	String	数字格式的表现类型，可以选择货币、百分比和数字类型。详细类型如下表所示	可以
pattern	String	模式字符串	可以
parseLocale	String，java.util.Locale	指定不同的国家区域	可以
var	String	存储格式化后的数字格式	不可以
scope	String	变量的作用范围	不可以
integerOnly	boolean	是否只显示整数部分	可以

type 属性的类型表

类　型	说　明	示　例
number	数字格式	6.7
currency	货币格式	￥6.70
percent	百分比格式	670%

示例

应用<fmt:parseNumber>标签将百分比格式、货币格式和数字格式的字符解析成数字类型，并进行算术运算。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<fmt:parseNumber value="25%" var="num" type="percent"/>
${num*3}            <!-- 显示 0.75 -->
<br>
<fmt:parseNumber value="￥3.14" var="num" type="currency"/>
${num*3}            <!-- 显示 9.42 -->
<fmt:parseNumber value="80,000" var="num"/>
${num+100}          <!-- 显示 80100 -->
```

示例

将以科学计数法表示的字符串解析成数字类型，并进行算术运算。

```
<fmt:parseNumber value="3.142E6" var="num" pattern="0.###E0"/>
${num*2 }   <!-- 显示 6284000 -->
<fmt:parseNumber value="31.416E-2" var="num" pattern="00.###E0"/>
${num*2 }   <!-- 显示 0.62832 -->
```

示例

测试 integerOnly 属性的用法。

```
<fmt:parseNumber value="200,233.56" var="num" integerOnly="true"/>
${num}                    <!-- 显示 200233 -->
<fmt:parseNumber value="200,233.56" var="num" integerOnly="false"/>
${num}                    <!-- 显示 200233.56 -->
```

10.3 <fmt:formatDate>标签——格式化时间

<fmt:formatDate>标签可以把字符串类型的数字解析成数字类型的数值，使其可以组合算术运算形成其他数值结果。

语法

```
<fmt:formatDate value="date"
[type="{time|date|both}"]
[dateStyle="{default|short|medium|long|full}"]
[timeStyle="{default|short|medium|long|full}"]
[pattern="customPattern"]
[timeZone="timeZone"]
[var="varName"]
[scope="{page|request|session|application}"]/>
```

<fmt:formatDate>标签的各属性的详细介绍如下表所示。

<fmt:formatDate>标签属性表

属　性	类　型	描　述	引用 EL
value	java.util.Date	将要格式化的日期对象	可以
type	String	指定显示日期对象的日期还是时间，可以两者都显示，它的取值可选 time 时间、date 日期和 both 两者	可以
pattern	String	模式字符串	可以
dateStyle	String	指定日期的显示方式	可以
timeStyle	String	指定时间的显示方式	可以
timeZone	String，java.util.TimeZone	设置时区	可以
var	String	存储格式化后日期对象的格式	不可以
scope	String	变量的作用范围	不可以

示例

应用<fmt:formatDate>标签格式化当前日期。

```
<%@ page import="java.util.Date" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%request.setAttribute("date",new Date()); %>
<!-- 显示 2008-11-29 11:28:12    -->
```


PDF电子书基地 http://dayo1982.400gb.com

```
<fmt:formatDate value="${date}" type="both"/>
<!-- 显示 2008 年 11 月 29 日 星期六 上午 11 时 28 分 12 秒 CST  -->
<fmt:formatDate value="${date}" dateStyle="full" timeStyle="full" type="both"/>
<fmt:formatDate value="${date}" type="date"/><!-- 显示 2008-11-29 -->
<!-- 显示 2008 年 11 月 29 日 -->
<fmt:formatDate value="${date}" dateStyle="long" type="date"/>
<!-- 显示 08-11-29  -->
<fmt:formatDate value="${date}" dateStyle="short" type="date"/>
<fmt:formatDate value="${date}" type="time"/>        <!-- 显示 11:28:12 -->
<!-- 显示午 11 时 28 分 12 秒 -->
<fmt:formatDate value="${date}" timeStyle="long" type="time"/>
<!-- 显示上午 11:28    -->
<fmt:formatDate value="${date}" timeStyle="short" type="time"/>
```

示例

以北京时间 2008-11-29 11:35:19 为例，测试 timeZone 属性。

```
<%@ page import="java.util.Date" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%request.setAttribute("date",new Date()); %>
<!-- 显示 2008-11-28 19:35:19    -->
<fmt:formatDate value="${date}" type="both"
timeZone="America/Los_Angeles"/>
<!-- 显示 2008-11-29 11:35:19  -->
<fmt:formatDate value="${date}" type="both" />
```

说明

America/Los_Angeles 表示美国太平洋时区。

示例

按照自定义的格式输出当前日期时间。

```
<%@ page import="java.util.Date" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%request.setAttribute("date",new Date()); %>
<!-- 显示 20081129 114852 -->
<fmt:formatDate value="${date}" type="both" pattern="yyyyMMdd HHmmss" />
<!-- 显示 2008/11/29 -->
<fmt:formatDate value="${date}" type="date" pattern="yyyy/MM/dd" />
```

10.4 <fmt:parseDate>标签——转换为时间

<fmt:parseDate>标签用于解析字符串为日期对象，被解析的字符串可以指定日期模式来灵活地表达日期对象。

语法 1

```
<fmt:parseDate value="dateString"
[type="{time|date|both}"]
[dateStyle="{default|short|medium|long|full}"]
```

```
[timeStyle="{default|short|medium|long|full}"]
[pattern="customPattern"]
[timeZone="timeZone"]
[parseLocale="parseLocale"]
[var="varName"]
[scope="{page|request|session|application}"]/>
```

语法 2

```
<fmt:parseDate [type="{time|date|both}"]
[dateStyle="{default|short|medium|long|full}"]
[timeStyle="{default|short|medium|long|full}"]
[pattern="customPattern"]
[timeZone="timeZone"]
[parseLocale="parseLocale"]
[var="varName"]
[scope="{page|request|session|application}"]>
date value to be parsed
</fmt:parseDate>
```

<fmt:parseDate>标签的各属性的详细介绍如下表所示。

<fmt:parseDate>标签属性表

属　性	类　型	描　述	引用 EL
value	Java.util.Date	将要格式化的日期对象	可以
type	String	指定解析字符串中的日期还是时间，可以两者都显示，它的取值可选 time 时间、date 日期和 both 两者	可以
pattern	String	日期字符串的样式	可以
dateStyle	String	指定日期的解析方式	可以
timeStyle	String	指定时间的解析方式	可以
timeZone	String，java.util.TimeZone	设置时区	可以
parseLocale	String，java.util.Locale	设置国家区域属性	可以
var	String	存储解析后日期对象的格式	不可以
scope	String	变量的作用范围	不可以

示例

应用<fmt:parseDate>标签将符合日期格式的字符串解析为日期对象。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!--显示 Sat Nov 29 11:48:52 CST 2008 -->
<fmt:parseDate value="20081129 114852" var="dateTime" pattern="yyyyMMdd HHmmss" />
${dateTime}
<fmt:parseDate value="2008-11-29" var="dateTime" />
${dateTime}          <!--显示 Sat Nov 29 00:00:00 CST 2008 -->
```


PDF电子书基地 http://dayo1982.400gb.com

10.5 <fmt:setTimeZone>标签——设置时区

<fmt:setTimeZone>标签用于设置默认时区，也可以将设置的时区存储在scope 属性指定范围的变量中。

语法

```
<fmt:setTimeZone value="timeZone"
[var="varName"]
[scope="{page|request|session|application}"]/>
```

<fmt:setTimeZone>标签的各属性的详细介绍如下表所示。

<fmt:setTimeZone>标签属性表

属　性	类　型	描　述	引用 EL
value	String，Java.util.TimeZone	指定的时区	可以
var	String	存储时区的变量	不可以
scope	String	变量的存储范围	不可以

示例

应用<fmt:setTimeZone>标签设置默认时区为 GMT。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:setTimeZone value="GMT" scope="session"/>
<fmt:parseDate value="2008-11-29" var="dateTime" />
<br>
${dateTime}
```

10.6 <fmt:timeZone>标签——格式化时区

<fmt:timeZone>标签用于设置标签体内部的时区。应用该标签后，标签体内所有时间和日期都采用标签设置的时区，但它不会影响到标签外的时区设置。

语法

```
<fmt:timeZone value="timeZone">
body content
</fmt:timeZone>
```

<fmt:timeZone>标签的各属性的详细介绍如下表所示。

<fmt:timeZone>标签属性表

属　性	类　型	描　述	引用 EL
value	String，Java.util.TimeZone	指定的时区	可以

示例

应用<fmt:timeZone>标签设置标签体内的时区为 GMT。

```
<fmt:timeZone value="GMT">
<fmt:parseDate value="2008-11-29 00:00:00" var="dateTime" />
<br>
${dateTime}
</fmt:timeZone>
<fmt:parseDate value="2008-11-29 00:00:00" var="dateTime" />
<br>
${dateTime}
```

运行上面的代码将显示以下内容：

```
Sat Nov 29 08:00:00 CST 2008
Sat Nov 29 00:00:00 CST 2008
```

从该运行结果可以知道，标签体内与标签体外采用的不是同一时区。

10.7　<fmt:setBundle>标签——指定消息资源

<fmt:setBundle>标签用于读取绑定的消息资源文件，当 JSP 页面读取本地消息文本时，将从绑定的消息资源文件中读取相应的键值。

语法

```
<fmt:setBundle basename="basename"
[var="varName"]
[scope="{page|request|session|application}"]/>
```

<fmt:setBundle>标签的各属性的详细介绍如下表所示。

<fmt:setBundle>标签属性表

属　性	类　型	描　述	引用 EL
basename	String	指定消息资源文件的名称，不需要指定文件的扩展名称	可以
var	String	存储资源的变量	不可以
scope	String	变量的存储范围	不可以

说明

当 basename 为 null、空或是找不到资源文件时，在网页上会产生"???<key>???"的错误信息。

示例

应用<fmt:setBundle>标签读取本地消息文本。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:setBundle basename="localMessage" scope="page"/>
<fmt:message key="company" var="company"/>
<fmt:message key="author" var="author"/>
<fmt:message key="branch" var="branch"/>
公司：${company}
作者：${author}
部门：${branch}
```


PDF电子书基地 http://dayo1982.400gb.com

说明

localMessage.properties 文件位于 class 文件夹中。

localMessage.properties 文件的具体内容如下：

```
company=mrsoft
author=wgh
branch=Java Web
```

本实例的运行结果如下。

```
公司：mrsoft 作者：wgh 部门：Java Web
```

10.8 <fmt:bundle>标签——读取绑定消息资源

<fmt:bundle>标签用于读取绑定的消息资源文件，该标签只对标签体之内的范围有效。

语法

```
<fmt:bundle basename="basename"
[prefix="prefix"]>
body content
</fmt:bundle>
```

<fmt:setBundle>标签的各属性的详细介绍如下表所示。

<fmt:setBundle>标签属性表

属　性	类　型	描　述	引用 EL
basename	String	指定消息资源文件的名称，不需要指定文件的扩展名称	可以
prefix	String	前缀字符串	可以

说明

当 basename 为 null、空或是找不到资源文件时，在网页上会产生"???<key>???"的错误信息。

示例

应用<fmt:bundle>标签读取本地消息文本。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:bundle basename="localMessage" prefix="mr">
<fmt:message key="company" var="company"/>
<fmt:message key="author" var="author"/>
<fmt:message key="branch" var="branch"/>
公司：${company}
作者：${author}
部门：${branch}
</fmt:bundle>
```

说明

localMessage.properties 文件位于 class 文件夹中。

localMessage.properties 文件的具体内容如下。

```
mrcompany=mrsoft
mrauthor=wgh
mrbranch=Java Web
```

本实例的运行结果如下。

```
公司：mrsoft 作者：wgh 部门：Java Web
```

10.9　<fmt:message>标签——读取本地消息

<fmt:message>标签负责读取本地消息资源，它从指定的消息文本资源中读取对应的键值，并且可以将键值存储在指定范围的变量中。

语法 1

```
<fmt:mssage key="messageKey"
[bundle="resourceBundle"]
[var="varName"]
[scope="{page|request|session|application}"]/>
```

语法 2

```
<fmt:mssage key="messageKey"
[bundle="resourceBundle"]
[var="varName"]
[scope="{page|request|session|application}"]>
<fmt:param> subtags
</fmt:mssage>
```

语法 3

```
<fmt:mssage [bundle="resourceBundle"]
[var="varName"]
[scope="{page|request|session|application}"]>
key
optional <fmt:param> subtags
</fmt:mssage>
```

<fmt:message>标签的各属性的详细介绍如下表所示。

<fmt:message>标签属性表

属　性	类　型	描　述	引用 EL
key	String	指定键的名称	可以
bundle	LocalizationContext	指定消息文本的来源	可以
var	String	存储资源的变量	不可以
scope	String	变量的存储范围	不可以

PDF电子书基地 http://dayo1982.400gb.com

示例

应用<fmt:bundle>标签和<fmt:message>标签读取本地消息文本。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:bundle basename="localMessage">
<fmt:message key="company"/>
<fmt:message key="author"/>
<fmt:message key="branch"/>
</fmt:bundle>
```

说明

localMessage.properties 文件位于 class 文件夹中。

localMessage.properties 文件的具体内容如下：

```
company=mrsoft
author=wgh
branch=Java Web
```

本实例的运行结果如下。

```
mrsoft wgh Java Web
```

示例

应用<fmt:bundle>标签和<fmt:message>标签读取本地消息文本，并将读取的结果保存到变量中。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:bundle basename="localMessage">
<fmt:message key="company" var="company"/>
<fmt:message key="author" var="author"/>
<fmt:message key="branch" var="branch"/>
公司：${company}
作者：${author}
部门：${branch}
</fmt:bundle>
```

说明

localMessage.properties 文件位于 class 文件夹中。

localMessage.properties 文件的具体内容如下：

```
company=mrsoft
author=wgh
branch=Java Web
```

本实例的运行结果如下。

```
公司：mrsoft 作者：wgh 部门：Java Web
```

10.10 <fmt:param>标签——指定资源参数

<fmt:param>标签主要用于为<fmt:message>标签读取的消息资源指定参数值

（如果消息资源有参数）。它的使用很简单，只需要指定参数值便可。

语法 1

```
<fmt:param value="messageParameter"/>
```

语法 2

```
<fmt:param>
        body content
</fmt:param>
```

<fmt:param>标签只有一个 value 属性，用于为消息文本传递参数值，该属性值可以引用 EL。

示例

读取带参数的本地消息资源，并通过<fmt:param>标签指定参数值。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:bundle basename="localMessage">
<fmt:message key="company"/>
<fmt:message key="author">
    <fmt:param value="sk"/>
</fmt:message>
<fmt:message key="date">
    <fmt:param value="<%=new Date() %>"/>
</fmt:message>
</fmt:bundle>
```

说明

localMessage.properties 文件位于 class 文件夹中。

localMessage.properties 文件的具体内容如下：

```
company=mrsoft
author={0}
date={0}
```

本实例的运行结果如下：

```
mrsoft sk 08-11-29 下午 4:48
```

10.11 <fmt:setlocale>标签——设置语言区域

<fmt:setlocale>标签主要用于设置语言区域。

语法

```
<fmt:setLocale value="locale"
[variant="variant"]
[scope="{page|request|session|application}"]/>
```

<fmt:setlocale>标签的各属性的详细介绍如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

<fmt:setlocale>标签属性表

属　性	类　型	描　述	引用 EL
value	String	指定区域代码。例如 zh_CN	可以
variant	String	存储时区的变量	可以
scope	String	变量的存储范围	不可以

10.12 <fmt:requestEncoding>标签——设置编码

<fmt:requestEncoding>标签主要用于设置请求的编码格式。

语法

```
<fmt:requestEncoding [value="charsetName"]/>
```

<fmt:requestEncoding>标签只有一个可选的 value 属性，用于以字符串的形式指定编程格式，如果没有指定该属性，标签会尝试自动查询编码格式。

示例

设置请求的编码格式为 GBK。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:requestEncoding value="GBK"/>
设置 GBK 编码时的输出结果：
${param.username }
```

如果输入的用户名字符串为“明日科技”，运行上面的代码，将显示如图 10.1 所示的内容。

设置GBK编码时的输出结果：明日科技

图 10.1　运行结果

注意

使用<fmt:requestEncoding>标签时，一定要在所有获取请求参数值的代码之前。当一个页面中同时应用了两个该标签，则以后面的标签为主。

示例

设置请求的编码格式为 ISO-8859-1。

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:requestEncoding value="ISO-8859-1"/>
设置 ISO-8859-1 编码时的输出结果：
${param.username }
```

如果输入的用户名字符串为“明日科技”，运行上面的代码，将显示如图 10.2 所示的内容。

```
设置ISO-8859-1编码时的输出结果： ?÷??????
```

图 10.2　运行结果

如果输入的用户名字符串为 mrsoft，运行上面的代码，将显示如图 10.3 所示的内容。

```
设置ISO-8859-1编码时的输出结果： mrsoft
```

图 10.3　运行结果

PDF电子书基地 http://dayo1982.400gb.com

第 11 章　JSTL 的 SQL 标签库

11.1　<sql:setDataSource>标签——设置数据源

<sql:setDataSource>标签用于设置数据源。数据源包括数据库的驱动、连接数据库的用户名、密码和 URL 连接等属性。

说明

在应用<sql:setDataSource>标签时，需要提供对应数据库的驱动包。

语法

```
<sql:setDataSource
{dataSource="dataSource" |
url="jdbcUrl"
[driver="driverClassName"]
[user="userName"]
[password="password"]}
[var="varName"]
[scope="{page|request|session|application}"]/>
```

<sql:setDataSource>标签的各属性的详细介绍如下表所示。

<sql:setDataSource>标签属性表

属　性	类　型	描　述	引用 EL
driver	String	连接数据库的 JDBC 驱动类名称	可以
dataSource	String，javax.sql.DataSource	使用其他已存在的数据源对象	可以
url	String	JDBC 连接数据库的 URL 地址	可以
user	String	连接数据库的用户名	可以
password	String	连接数据库的密码	可以
var	String	存储数据源的变量	不可以
scope	String	变量的存储范围	不可以

说明

JSTL 的 SQL 标签无法实现数据连接池等高级数据源功能。

示例

应用<sql:setDataSource>标签连接 SQL Server 2000 数据库 db_testJSTL。

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
连接 SQL Server 2000 数据库
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testJSTL"
 user="sa"
 password=""/>
```

示例

应用<sql:setDataSource>标签连接 MySQL 数据库 db_testJSTL。

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
连接 MySQL 数据库
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
```

11.2　<sql:query>标签——通过 SQL 语句查询

<sql:query>标签用于通过 SQL 语句查询符合条件的数据。

语法 1

```
<sql:query sql="sqlQuery"
var="varName" [scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]/>
```

语法 2

```
<sql:query sql="sqlQuery"
var="varName" [scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
<sql:param> actions
</sql:query>
```

语法 3

```
<sql:query var="varName"
[scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
query
optional <sql:param> actions
</sql:query>
```

<sql:query>标签的各属性的详细介绍如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

<sql:query>标签属性表

属　性	类　型	描　述	引用 EL
sql	String	查询数据的 SQL 语句	可以
DataSource	String，javax.sql.DataSource	使用的数据源对象	可以
maxRows	String	指定查询记录的最大行数	可以
startRow	String	指定查询记录的开始行数	可以
var	String	存储查询结果的变量，该变量包含 5 个操作结果集的属性，相应的属性说明如下表所示	不可以
scope	String	变量的存储范围	不可以

说明

dataSource 属性是可选项，如果没有设置该属性，会在 page 范围内寻找数据源对象，如果找不到，则会抛出 JspException 异常。

结果集变量的属性表

属　性	类　型	描　述
rowCount	int	结果集中的记录总数
columnNames	String[]	字段名称数组
rows	java.util.Map	以字段名索引结果集
rowsByIndex	Object[]	以数字下标索引结果集
limitedByMaxRows	boolean	是否设置了 maxRows 属性来限制查询记录的数量

示例

应用<sql:setDataSource>标签连接 SQL Server 2000 数据库 db_testJSTL，并应用<sql:query>标签输出数据表 tb_user 中的从第 3 条记录开始的 5 条记录。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testJSTL"
 user="sa" password=""/>
    <sql:query var="rs" sql="SELECT * FROM tb_user" maxRows="5" startRow="2"/>
    <table width="300" border="1" cellspacing="0" cellpadding="0"  bordercolor="#FFFFFF"
bordercolordark="#999999" bordercolorlight="#FFFFFF">
      <tr>
        <td align="center"> ${rs.columnNames[0]}</td>
        <td align="center"> ${rs.columnNames[1]}</td>
        <td align="center"> ${rs.columnNames[2]}</td>
```

```
    </tr>
      <c:forEach items="${rs.rowsByIndex}" var="row">
    <tr>
      <td> ${row[0]}</td>
      <td> ${row[1]}</td>
      <td> ${row[2]}</td>
    </tr>
      </c:forEach>
</table>
```

示例

应用<sql:setDataSource>标签连接 MySQL 数据库 db_testJSTL，并应用<sql:query>标签输出数据表 tb_user 中的全部记录。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
连接 MySQL 数据库
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
<sql:query var="rs" sql="SELECT * FROM tb_user"/>
<table width="300" border="1" cellspacing="0" cellpadding="0"   bordercolor="#FFFFFF"
bordercolordark="#999999" bordercolorlight="#FFFFFF">
  <tr>
    <td align="center"> ${rs.columnNames[0]}</td>
    <td align="center"> ${rs.columnNames[1]}</td>
    <td align="center"> ${rs.columnNames[2]}</td>
  </tr>
    <c:forEach items="${rs.rowsByIndex}" var="row">
  <tr>
    <td> ${row[0]}</td>
    <td> ${row[1]}</td>
    <td> ${row[2]}</td>
  </tr>
    </c:forEach>
</table>
```

示例

在<sql:query>标签中指定数据源为<sql:setDataSource>标签创建的数据源 conn。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testJSTL"
 user="sa" password="" var="conn"/>
```


PDF电子书基地 http://dayo1982.400gb.com

```
<sql:transaction dataSource="${conn}">
<sql:query var="rs" sql="SELECT * FROM tb_user"/>
```

11.3 <sql:update>标签——更新数据

<sql:update>标签用于使用 update、delete 和 insert 等 SQL 语句更新数据库记录，并返回影响的记录行数。

语法 1

```
<sql:update sql="sqlUpdate"
[dataSource="dataSource"]
[var="varName"] [scope="{page|request|session|application}"]/>
```

语法 2

```
<sql:update sql="sqlUpdate"
[dataSource="dataSource"]
[var="varName"] [scope="{page|request|session|application}"]>
<sql:param> actions
</sql:update>
```

语法 3

```
<sql:update [dataSource="dataSource"]
[var="varName"] [scope="{page|request|session|application}"]>
update statement
optional <sql:param> actions
</sql:update>
```

<sql:update>标签的各属性的详细介绍如下表所示。

<sql:update>标签属性表

属　性	类　型	描　述	引用 EL
sql	String	更新数据记录的 SQL 语句	可以
dataSource	String，javax.sql.DataSource	使用的数据源对象	可以
var	String	存储更新记录数量的变量	不可以
scope	String	变量的存储范围	不可以

示例

应用<sql:setDataSource>标签连接 MySQL 数据库 db_testJSTL，并应用<sql:update>标签向数据表 tb_user 中插入一条记录，并显示全部数据表记录。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
```

```
    <sql:update sql="INSERT INTO tb_user (username,pwd) VALUES('无语','111')"/>
    <table width="300" border="1" cellspacing="0" cellpadding="0" bordercolor="#FFFFFF"
bordercolordark="#999999" bordercolorlight="#FFFFFF">
      <tr>
        <td align="center"> ${rs.columnNames[0]}</td>
        <td align="center"> ${rs.columnNames[1]}</td>
        <td align="center"> ${rs.columnNames[2]}</td>
      </tr>
      <c:forEach items="${rs.rowsByIndex}" var="row">
    <tr>
      <td> ${row[0]}</td>
      <td> ${row[1]}</td>
      <td> ${row[2]}</td>
    </tr>
      </c:forEach>
  </table>
```

示例

应用<sql:setDataSource>标签连接 MySQL 数据库 db_testJSTL，并应用<sql:update>标签修改数据表 tb_user 中用户名为“无语”的用户密码为 123。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
<sql:update sql="UPDATE tb_user SET pwd='123' WHERE username='无语'"/>
<sql:query var="rs" sql="SELECT * FROM tb_user order by id desc"></sql:query>
```

示例

应用<sql:setDataSource>标签连接 MySQL 数据库 db_testJSTL，并应用<sql:update>标签删除数据表 tb_user 中用户名为“无语”的用户信息。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
<sql:update sql="DELETE FROM tb_user WHERE username='无语'"/>
```

11.4　<sql:param>标签——指定 SQL 参数

<sql:param>标签用于动态地为 SQL 语句指定参数值，这不同于普通的以变量填充参数的方式，使用<sql:param>标签指定 SQL 参数值可以防止 SQL 注入。

PDF电子书基地 http://dayo1982.400gb.com

语法 1

```
<sql:param value="value"/>
```

语法 2

```
<sql:param>
parameter value
</sql:param>
```

<sql:param>标签只有一个 value 属性，用于为 SQL 语句指定参数值。

示例

应用<sql:update>标签修改数据表 tb_user 中指定用户的密码，其中，用户名和密码通过<sql:param>标签动态指定。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/db_testjstl" user="root"
 password="111"/>
<sql:update sql="UPDATE tb_user SET pwd=? WHERE username=?">
    <sql:param value="123"></sql:param>
    <sql:param value="小小"></sql:param>
</sql:update>
```

11.5 <sql:dateParam>标签——填充日期型参数

<sql:dateParam>标签和<sql:param>标签功能相似，不过<sql:dateParam>主要用于为 SQL 语句填充日期类型的参数值。

语法

```
<sql:dateParam value="value" [type="{timestamp|time|date}"]/>
```

<sql:dateParam>标签的各属性的详细介绍如下表所示。

<sql:dateParam>标签属性表

属　性	类　型	描　述	引用 EL
value	java.util.Date	日期参数值	可以
type	String	日期参数种类，其取值范围可以选择 time 时间类型、date 日期类型和 timestamp 日期与时间	可以

示例

查询注册日期为 2008-12-1 的用户信息。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testJSTL"
 user="sa" password=""/>
<%
Date date=new Date(new GregorianCalendar(2008,11,1).getTimeInMillis());
request.setAttribute("date",date);
%>
<sql:query var="rs" sql="SELECT * FROM tb_user WHERE createTime=?">
    <sql:dateParam value="${date}" type="date"/>
</sql:query>
```

示例

查询注册时间为 2008-12-1 15:24:12 的用户信息。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testJSTL"
 user="sa" password=""/>
<%
Date date=new Date(new GregorianCalendar(2008,11,1,15,24,12).getTimeInMillis());
request.setAttribute("date",date);
%>
<sql:query var="rs" sql="SELECT * FROM tb_user WHERE createTime=?">
    <sql:dateParam value="${date}" type="timestamp"/>
</sql:query>
```

11.6 <sql:transaction>标签——事务处理

<sql:transaction>标签用于在事务中处理 SQL 操作，如果 SQL 操作有错误将不会执行<sql:transaction>标签体中的所有 SQL 操作。

语法

```
<sql:transaction [dataSource="dataSource"]
[isolation=isolationLevel]>
<sql:query> and <sql:update> statements
</sql:transaction>
isolationLevel ::= "read_committed"
| "read_uncommitted"
| "repeatable_read"
| "serializable"
```

<sql:transaction>标签的各属性的详细介绍如下表所示。

PDF电子书基地 http://dayo1982.400gb.com



<sql:transaction>标签属性表

属　性	类　型	描　述	引用 EL
dataSource	String，javax.sql.DataSource	使用的数据源对象	可以
isolation	String	事务的隔离属性，用来设置事务的安全级别。其属性值包括 read_committed、read_uncommitted、repeatable_read 和 serializable	可以



示例

应用<sql:transaction>标签在事务中处理 SQL 操作。

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
 url="jdbc:microsoft:sqlserver://localhost:1433;databaseName=db_testH"
 user="sa" password="" var="conn"/>
<sql:transaction dataSource="${conn}" isolation="read_uncommitted">
<sql:update sql="UPDATE tb_user SET pwd=? WHERE username=?">
    <sql:param value="1224"></sql:param>
    <sql:param value="wgh"></sql:param>
</sql:update>
<sql:update sql="UPDATE tb_user SET id=? WHERE username=?">
    <sql:param value="123"></sql:param>
    <sql:param value="wgh"></sql:param>
</sql:update>
</sql:transaction>
```

运行上面的代码，如果第二条更新语句出现错误，则第一条更新语句的更新结果将会被回滚。

第 12 章　JSTL 的 XML 标签库

12.1　<x:parse>标签——解析指定的 XML 内容

<x:parse>标签可以解析指定的 XML 内容。

语法 1

```
<x:parse {doc="XMLDocument"|xml1="XMLDocument"}
{var="var" [scope="scope"]|varDom="var" [scopeDom="scope"]}
[systemId="systemId"]
[filter="filter"]/>
```

语法 2

```
<x:parse
{var="var" [scope="scope"]|varDom="var" [scopeDom="scope"]}
[systemId="systemId"]
[filter="filter"]>
XML Document to parse
</x:parse>
```

说明

<x:parse>标签有两种语法格式，这两种语法格式的输出结果完全相同。

<x:parse>标签的各属性的详细介绍如下表所示。

<x:parse>标签属性表

属　性	类　型	描　述	引用 EL
doc	String/Reader	提供解析的 XML 文件	可以
xml	String	同 doc 属性相同，已经不再使用	可以
var	String	存储解析后的 XML	不可以
varDom	String	以 org.w3c.dom.Document 类型存储解析后的 XML 文件	不可以
scope	String	var 属性的存储范围	不可以
scopeDom	String	varDom 属性的存储范围	不可以
systemId	String	XML 文件中的 URI 信息，用于解析 XML 文件的路径	可以
filter	org.xml.sax.XMLFilter	解析 XML 文件使用的过滤器	可以

示例

首先通过<c:import>标签导入 bookInfo.xml 文件，然后通过<x:parse>标签解

PDF电子书基地 http://dayo1982.400gb.com

析 bookInfo.xml 文件的内容，最后通过<x:out>标签输出 XML 文件中的部分内容。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
编号：<x:out select="$bookInfo/books/book/id/@value"/><br>
书名：<x:out select="$bookInfo/books/book/name/@value"/><br>
出版社：<x:out select="$bookInfo/books/book/publish/@value"/>
```

如果 bookInfo.xml 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<books>
    <book>
        <id value="1"/>
        <name value="JSP 程序开发范例宝典"/>
        <author value="王国辉"/>
        <publish value="人民邮电出版社"/>
    </book>
</books>
```

运行本实例时，将显示以下内容：

```
编号：1
书名：JSP 程序开发范例宝典
出版社：人民邮电出版社
```

12.2 <x:out>标签——输出 XML 信息

<x:out>标签和<c:out>标签类似，它们都是输出标签，<x:out>标签主要用于输出 XML 信息。

语法 1

```
<x:out select="XPathExpression" [escapeXml="{true|false}"]/>
```

语法 2

```
<x:out select="XPathExpression" [escapeXml="{true|false}"]>
</x:out>
```

说明

<x:out>标签有两种语法格式，这两种语法格式的输出结果完全相同。

<x:out>标签的各属性的详细介绍如下表所示。

<x:out>标签属性表

属　性	类　型	描　述	引用 EL
select	String	XPath 语句的表达式，用于查询指定节点的属性值	不可以
escapeXml	boolean	指定是否要转换特殊字符，这些特殊字符包括<、>、&、'和"。默认为 true	可以

XPath 表达式使用路径表示法寻址 XML 文档的各个部分，包括 XML 文档的根节点、父节点、子节点和兄弟节点等。常用的 XPath 表达式及对应的功能如下表所示。

XPath 表达式及对应功能

表 达 式	功 能
/	根节点
./	文件节点
../	父节点
节点名称	子节点
../结点名称	兄弟节点
//节点名称	所有同一节点
@属性名称	属性
*	全局

说明

在 XML 文件中，属性值或是元素内容中不能出现<、>、&、'和"等字符，当这些内容中必须包括这字符时，可以应用对应的实体代码表示。下面将通过表格的形式给出<、>、&、'和"等字符的实体代码。

特殊字符的实体代码

字 符	实 体 代 码	字 符	实 体 代 码
<	<	>	>
&	&	'	'
"	"	—	—

示例

首先通过<c:import>标签导入 bookInfo.xml 文件，然后通过<x:parse>标签解析 bookInfo.xml 文件的内容，最后通过<x:out>标签输出 XML 文件中的部分内容。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
编号：<x:out select="$bookInfo/books/book/id/@value"/><br>
书名：<x:out select="$bookInfo/books/book/name" escapeXml="true" /><br>
出版社：<x:out select="$bookInfo/books/book/publish/@value"/>
```

如果 bookInfo.xml 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<books>
    <book>
```


PDF电子书基地 http://dayo1982.400gb.com

```
        <id value="1"/>
        <name>&lt;&lt;JSP 程序开发范例宝典&gt;&gt;</name>
        <author value="明日科技"/>
        <publish value="人民邮电出版社"/>
    </book>
</books>
```

运行本实例时，将显示如图 12.1 所示的运行结果。

如果将示例中的以下代码

```
书名：<x:out select="$bookInfo/books/book/name" escapeXml="true" />
```

修改为

```
书名：<x:out select="$bookInfo/books/book/name" escapeXml="false" />
```

将显示如图 12.2 所示的运行结果。

```
编号：1
书名：<<JSP程序开发范例宝典>>
出版社：人民邮电出版社
```

图 12.1 运行结果

```
编号：1
书名：<>
出版社：人民邮电出版社
```

图 12.2 运行结果

12.3 <x:set>标签——保存 XML 节点

<x:set>标签用于把从 XML 文件指定节点读取的属性值存储到指定范围的变量中。

语法 1

```
<x:set select="XPathExpression"
var="varName" [scope="{page|request|session|application}"]/>
```

语法 2

```
<x:set select="XPathExpression"
var="varName" [scope="{page|request|session|application}"]>
</x:set>
```

说明

<x:set>标签有两种语法格式，这两种语法格式的输出结果完全相同。

<x:set>标签的各属性的详细介绍如下表所示。

<x:set>标签属性表

属　性	类　型	描　述	引用 EL
select	String	XPath 语句的表达式	不可以
var	String	将读取的 XML 信息存储值变量中	不可以
scope	String	var 属性的存储范围	不可以

示例

首先通过<c:import>标签导入 bookInfo.xml 文件，然后通过<x:parse>标签解析 bookInfo.xml 文件的内容，再通过<x:set>标签获取每本图书信息，并保存到变

量 book 中，最后通过<x:out>标签输出包括书名、出版社和单价的图书信息。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
<x:set var="book" select="$bookInfo/books/book"/>
书名：<x:out select="$book/name" escapeXml="true" /><br>
出版社：<x:out select="$book/publish/@value"/><br>
单价：<x:out select="$book/price/@value"/>
```

12.4 <x:if>标签——XPath 条件执行 JSP 代码

<x:if>标签根据 XPath 条件语句执行指定的 JSP 代码。

语法 1

将条件判断结果保存在 var 属性指定的变量中。

```
<x:if select="XPathExpression"
var="varName" [scope="{page|request|session|application}"]/>
```

语法 2

将根据条件表达式的判断结果去执行标签主体的 JSP 代码。

```
<x:if select="XPathExpression"
[var="varName"] [scope="{page|request|session|application}"]>
body content（标签体）
</x:if>
```

<x:if>标签的各属性的详细介绍如下表所示。

<x:if>标签属性表

属　性	类　型	描　述	引用 EL
select	String	XPath 语句的条件表达式，如果判断结果为 true，则执行标签体的 JSP 代码	不可以
var	String	将判断结果（true 或 false）保存到变量中	不可以
scope	String	var 属性的存储范围	不可以

示例

personnel.xml 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<personnel>
    <person>
        <name value="wgh"/>
        <knowledge>本科</knowledge>
        <memo value=""/>
    </person>
</personnel>
```

通过<x:if>标签判断 personnel.xml 文件中，memo 节点的值是否空，如果为

PDF电子书基地 http://dayo1982.400gb.com

空则显示“无简历”。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="personnel.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="personnel" doc="${xmlFile}"></x:parse>
<x:if select="$personnel/personnel/person/memo[@value='']">无简历</x:if>
```

通过<x:if>标签判断 personnel.xml 文件中，knowledge 节点的值是否空，如果不为空则显示“学历不为空”。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="personnel.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="personnel" doc="${xmlFile}"></x:parse>
<x:if select="$personnel/personnel/person[knowledge!='']">学历不为空</x:if>
```

12.5 <x:choose>标签——完成条件判断

<x:choose>标签与其子标签<x:when>和<x:otherwise>用于完成条件判断。

语法

```
<x:choose>
body content (<x:when> and <x:otherwise> subtags)
</x:choose>
```

示例

应用<x:choose>标签显示图书“JSP 程序开发范例宝典”的畅销程度。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
<x:choose>
    <x:when  select="$bookInfo/books/book[name='JSP 程序开发范例宝典'][sellAmount/@value>
5000]">非常畅销</x:when>
    <x:when  select="$bookInfo/books/book[name='JSP 程序开发范例宝典'][sellAmount/@value<
5000][sellAmount/@value>2000]">基本畅销</x:when>
    <x:otherwise>不畅销</x:otherwise>
</x:choose>
```

12.6 <x:when>标签——根据条件执行代码

<x:when>标签为<x:choose>标签的子标签，用于根据指定的条件执行不同的程序代码。

语法

```
<x:when select="XPathExpression">
body content（标签体）
</x:when>
```

<x:when>标签的各属性的详细介绍如下表所示。

<x:when>标签属性表

属　性	类　型	描　述	引用 EL
select	String	XPath 语句的条件表达式，如果判断结果为 true，则执行标签体的 JSP 代码	不可以

示例

应用<x:choose>标签和<x:when>标签显示图书“JSP 程序开发范例宝典”的畅销程度。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
<x:choose>
    <x:when select="$bookInfo/books/book[name='JSP 程序开发范例宝典'][sellAmount/@value>
5000]">非常畅销</x:when>
    <x:when select="$bookInfo/books/book[name='JSP 程序开发范例宝典'][sellAmount/@value<
5000][sellAmount/@value>2000]">基本畅销</x:when>
    <x:when select="$bookInfo/books/book[name='JSP 程序开发范例宝典'][sellAmount/@value
<2000]">不畅销</x:when>
</x:choose>
```

12.7 <x:otherwise>标签——不满足条件执行

<x:otherwise>标签是<x:choose>标签的默认执行标签，在没有满足条件的情况下会执行该标签体。

语法

```
<x:otherwise>
conditional block
</x:otherwise>
```

示例

应用<x:choose>标签、<x:when>标签和<x:otherwise>标签根据输入简历的情况，给予不同的提示。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="personnel.xml" var="xmlFile1" charEncoding="gbk"/>
<x:parse var="personnel" doc="${xmlFile1}"></x:parse>
<x:choose>
    <x:when select="$personnel/personnel/person[memo/@value='']">请输入简历！ </x:when>
    <x:otherwise>表现的很好，我们会仔细查看你的简历的！ </x:otherwise>
</x:choose>
```


PDF电子书基地 http://dayo1982.400gb.com

12.8 <x:forEach>标签——遍历 XML 文件

<x:forEach>标签用于根据提供的 XPath 表达式，遍历 XML 文件的内容。

语法

```
<x:forEach[var="varName"] select="XPathExpression">
[varStatus="varStatusName"]
[begin="begin"] [end="end"] [step="step"]>
body content
</x:forEach>
```

<x:forEach>标签的各属性的详细介绍如下表所示。

<x:forEach>标签属性表

属　性	类　型	描　述	引用 EL
select	String	XPath 表达式，决定查询结果	不可以
var	String	循环体的变量	不可以
begin	int	循环的起始位置，包括该位置	可以
end	int	循环的终止位置，包括该位置	可以
step	int	循环的步长	可以
varStatus	String	循环的状态变量	不可以

示例

遍历 XML 文件 bookInfo.xml 的全部图书信息。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
<x:forEach select="$bookInfo/books/book">
    <p><x:out select="id/@value"/><br>
    <x:out select="name"/><br>
    <x:out select="publish/@value"/><br>
    <x:out select="sellAmount/@value"/></p>
</x:forEach>
```

示例

遍历 XML 文件 bookInfo.xml，并显示第 1 本～第 3 本的图书信息。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="bookInfo.xml" var="xmlFile" charEncoding="gbk"/>
<x:parse var="bookInfo" doc="${xmlFile}"></x:parse>
<x:forEach select="$bookInfo/books/book" begin="0" end="2">
    <p><x:out select="id/@value"/><br>
    <x:out select="name"/><br>
    <x:out select="publish/@value"/><br>
```

```
        <x:out select="sellAmount/@value"/></p>
    </x:forEach>
```

12.9 <x:transform>标签——XML 到 XSLT 转换

<x:transform>标签用于完成 XML 到 XSLT 样式的转换。

说明

在使用<x:transform>标签时，需要导入 xalan.jar 和 serializer.jar 包，否则将抛出异常。

语法 1

```
<x:transform
{doc="XMLDocument"|xml1="XMLDocument"} xslt="XSLTStylesheet"
[{docSystemId="XMLSystemId"|xmlSystemId1="XMLSystemId"}]
[xsltSystemId="XSLTSystemId"]
[{var="varName" [scope="scopeName"]|result="resultObject"}]
```

语法 2

```
<x:transform
{doc="XMLDocument"|xml1="XMLDocument"} xslt="XSLTStylesheet"
[{docSystemId="XMLSystemId"|xmlSystemId1="XMLSystemId"}]
[xsltSystemId="XSLTSystemId"]
[{var="varName" [scope="scopeName"]|result="resultObject"}]
<x:param> actions
</x:transform>
```

语法 3

```
<x:transform
xslt="XSLTStylesheet"
[{docSystemId="XMLSystemId"|xmlSystemId1="XMLSystemId"}]
xsltSystemId="XSLTSystemId"
[{var="varName" [scope="scopeName"]|result="resultObject"}]
XML Document to parse
optional <x:param> actions
</x:transform>
```

<x:transform>标签的各属性的详细介绍如下表所示。

<x:transform>标签属性表

属　性	类　型	描　述	引用 EL
doc	String，Reader，javax.xml.transform. Source，org.w3c.dom.Document	XML 文件来源	可以
xslt	String，Reader，javax.xml.transf orm.Source	转换 XML 的样式模板	可以
docSystemId	String	分析 XML 文件的系统 ID	可以

PDF电子书基地 http://dayo1982.400gb.com

续表

属　性	类　型	描　述	引用 EL
result	javax.xml.transform.Result	存储转换后的结果对象	可以
var	String	以 rg.w3c.dom.Document 类型存储转换后的结果	不可以
scope	String	var 属性的范围	不可以
xsltSystemId	String	分析 XSLT 样式的系统 ID	可以

示例

将 XML 文件 personnel.xml 套用 XSLT 文件 personnel.xsl 样式进行输出。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="personnel.xml" var="xmlFile" charEncoding="gbk"/>
<c:import url="personnel.xsl" var="xsltFile" charEncoding="gbk"/>
<x:transform doc="${xmlFile}" xslt="${xsltFile}"/>
```

personnel.xsl 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<xsl:stylesheet version=
    "1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
    <xsl:template match="/" >
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="personnel">
        <html>
            员工信息<br/>
            <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="name">
        <xsl:value-of select="@value"/>
    </xsl:template>
    <xsl:template match="knowlodge">
        <xsl:value-of select="."/>
    </xsl:template>
    <xsl:template match="memo">
        <xsl:value-of select="@value"/><br/>
    </xsl:template>
</xsl:stylesheet>
```

personnel.xml 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<personnel>
    <person>
        <name value="wgh"/>
        <knowledge>本科</knowledge>
        <memo value="计划要注重实效性"/>
```

```
        </person>
        <person>
            <name value="sk"/>
            <knowledge>本科</knowledge>
            <memo value="诚信"/>
        </person>
</personnel>
```

运行结果如图 12.3 所示。

```
员工信息
wgh 本科 计划要注重实效性
sk 本科 诚信
```

图 12.3 运行结果

12.10 <x:param>标签——定义参数

<x:transform>标签用于为<x:transform>标签设定参数信息。

如果执行文件转换的样式表使用了参数，可以使用<x:param>标签来定义这些参数。

语法 1

```
<x:param name="name" value="value"/>
```

语法 2

```
<x:param name="name">
parameter value
</x:param>
```

<x:param>标签的各属性的详细介绍如下表所示。

<x:param>标签属性表

属　性	类　型	描　述	引用 EL
name	String	用于指定参数名	可以
value	Object	用于设置参数值	可以

示例

将 XML 文件 personnel.xml 套用 XSLT 文件 personnel.xsl 样式，并输出<name>节点不等于由<x:param>标签指定的参数 paraName 的值的用户信息。

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:import url="personnel.xml" var="xmlFile" charEncoding="gbk"/>
<c:import url="personnel.xsl" var="sxsltFile" charEncoding="gbk"/>
<x:transform doc="${xmlFile}" xslt="${xsltFile}">
    <x:param name="paraName" value="wgh"></x:param>
</x:transform>
```


PDF电子书基地 http://dayo1982.400gb.com

personnel.xsl 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<xsl:stylesheet version=
    "1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
    <xsl:param name="paraName"/>
    <xsl:template match="/" >
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="personnel">
        <html>
            员工信息<br/>
            <xsl:apply-templates/>
        </html>
    </xsl:template>
    <xsl:template match="person">
        <xsl:if test="name/@value!=$paraName">
            <div style="font-size:9pt;">
            <pre>
            <xsl:value-of select="name/@value"/> | <xsl:value-of
select="knowledge"/> | <xsl:value-of select="memo/@value"/>
            </pre>
            </div>
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

personnel.xml 文件的内容如下：

```
<?xml version="1.0" encoding="gbk"?>
<personnel>
    <person>
        <name value="wgh"/>
        <knowledge>本科</knowledge>
        <memo value="计划要注重实效性"/>
    </person>
    <person>
        <name value="sk"/>
        <knowledge>本科</knowledge>
        <memo value="诚信"/>
    </person>
</personnel>
```

运行结果如图 12.4 所示。

员工信息

sk | 本科 | 诚信

图 12.4 运行结果

第 13 章　Struts 控制器组件

13.1　Action 类——请求处理

org.apache.struts.action.Action 类是用户请求和业务逻辑之间的桥梁。每个 Action 充当一项业务代理。

示例

本示例创建类 MyAction，该类继承自 Action 类。

```
public class MyAction extends Action {
                ……
}
```

13.1.1　addErrors 方法——添加指定的错误键

添加指定的错误键，可以通过<html:errors>标签获取错误提示。

方法声明

```
protected void addErrors(javax.servlet.http.HttpServletRequest request,
                         ActionMessages errors)
```

入口参数

request：HTTP 请求对象。

errors：ActionMessages 实例，每个 ActionMessages 实例都对应一条具体的错误信息。

示例

在 Action 实现类的 addErrors()方法中读取资源文件 key 为“users.null”对应的文本内容，将其保存在 request 对象中。

```
protected void addErrors(HttpServletRequest request, ActionMessages errors) {
    ActionError error = null;
    error = new ActionError("users.null"); //获取本地资源文件信息
    errors.add("userName", error);
    saveErrors(request, errors);
}
```

13.1.2　addMessages 方法——添加 ActionMessages

检查 request 对象中是否有 ActionMessages 对象，如果有，则将接受的 ActionMessages 保存到 request 中，如果没有，用 setAttribute()方法将 ActionMessages

PDF电子书基地 http://dayo1982.400gb.com

对象添加进去。

方法声明

```
protected void addMessages(javax.servlet.http.HttpServletRequest request,
                           ActionMessages errors)
```

入口参数

request：HTTP 请求对象。

errors：ActionMessages 实例，每个 ActionMessages 实例都对应一条具体的错误信息。

示例

在 Action 实现类的 addMessages()方法中读取资源文件 key 为“users.null”对应的文本内容，再调用 saveMessages()方法，把 ActionMessages 集合存放到 request 范围内。

```
protected void addMessages(HttpServletRequest request,
        ActionMessages errors) {
    errors.add("username", new ActionMessage("users.null"));
    saveMessages(request, errors);
}
```

13.1.3 execute 方法——完成用户请求处理

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

方法声明

```
public ActionForward execute(ActionMapping mapping , ActionForm form,
                             javax.servlet.http.HttpServletRequest request,
                             javax.servlet.http.HttpServletResponse response)
    throws java.lang.Exception
```

入口参数

mapping：用于描述控制器请求的 ActionMapping 实例。

form：用于保存用户请求表单中数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。

抛出异常：Exception。

示例

在 Action 实现类的 execute()方法中处理请求转发的地址。当访问此 Action 后，将访问相应页面。

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {
        return mapping.findForward("welcome");
}
```

13.1.4 isTokenValid 方法——验证令牌机制

Struts 中提供了令牌机制来解决 Web 应用中重复提交的问题。isTokenValid() 用来判断存储在当前用户会话中的令牌值和请求参数中的令牌值是否匹配。

方法声明 1

```
protected boolean idTokenValid(
        javax.servlet.http.HttpServletRequest request)
```

入口参数：request 是 HTTP 请求对象。

返回值：如果匹配，则返回 true，否则返回 false。

示例

在 execute()方法中，通过 isTokenValid()方法判断用户是否重复提交表单，并转发至相应页面。

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
            throws Exception {
     ActionMessages errors = new ActionMessages();
        if (!isTokenValid(request)) {
            errors.add(ActionMessages.GLOBAL_MESSAGE,
              new ActionMessage("error.invalid.token"));
            saveMessages(request, errors);
            saveToken(request);
            return new ActionForward(mapping.getInput());
        } else {
            resetToken(request);
        }
        return mapping.findForward("welcome");
    }
```

方法声明 2

检查令牌是否有效，并重置令牌。该方法主要防止用户由于网速慢，或者其他的原因不断刷新，造成表单重复提交而设计。

```
protected boolean idTokenValid(
        javax.servlet.http.HttpServletRequest request,boolean reset)
```

入口参数

request：HTTP 请求对象。

reset：如果重置，则为 true。

返回值：未重复提交时，则返回 true，否则返回 false。

PDF电子书基地 http://dayo1982.400gb.com





示例

通过 isTokenValid()方法判断用户是否重复提交表单。

```
if (isTokenValid(request,true)) {
    //未重复提交，实现代码
}
```

13.1.5 getResources 方法——获取默认 MessageResources

该方法获取默认的 MessageResources 对象。



方法声明 1

```
protected MessageResources getResources(
        javax.servlet.http.HttpServletRequest request)
```

入口参数：request 是 HTTP 请求对象。

返回值：封装 Resource Bundle 中文本内容的 MessageResources 对象。



示例

通过 getResources()方法获取 MessageResources 对象，读取信息 key 为“users.null”对应的文本内容，并将其输出。

```
MessageResources messge = getResources(request);
System.out.println(messge.getMessage("users.null"));
```



方法声明 2

一个 Struts 应用可以有多个资源文件。可以通过该方法获取特定的 Message 对象。

```
protected MessageResources getResources(
        javax.servlet.http.HttpServletRequest request,
        java.lang.String key)
```

入口参数

request：HTTP 请求对象。

key：与 Struts 配置文件的<message-resources>元素的 key 属性对应。

返回值：封装 Resource Bundle 中文本内容的 MessageResources 对象。



示例

通过 getResources()方法获取与 Struts 配置文件 key 值为“ap”对应的 MessageResources 对象，读取资源文件中 key 为“users.null”对应的文本内容输出。

```
MessageResources messge = getResources(request,"ap");
System.out.println(messge.getMessage("users.null"));
```

13.1.6 resetToken 方法——删除令牌属性

该方法从当前 session 范围内删除令牌属性。

方法声明

```
protected void resetToken(
        javax.servlet.http.HttpServletRequest request)
```

入口参数：request 是 HTTP 请求对象。

示例

在 execute()方法中，通过 isTokenValid()方法判断用户是否重复提交表单，并转发至相应页面。

```
if (!isTokenValid(request)) {
    //如果没有重复提交表单执行代码。
} else {
    resetToken(request);          //重置表单
}
```

13.1.7 saveToken 方法——保存新令牌

创建一个新的令牌，并把他保存在 session 范围内。如果 HttpSession 对象不存在，就先创建一个 HttpSession 对象。

方法声明

```
protected void saveToken(
        javax.servlet.http.HttpServletRequest request)
```

入口参数：request 是 HTTP 请求对象。

示例

在 execute()方法中，创建一个新的令牌。

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception {
    ActionMessages errors = new ActionMessages();
    saveToken(request);
    return mapping.findForward("welcome");
}
```

13.1.8 saveErrors 方法——保存错误信息

该方法把 ActionMessages 对象保存在 request 范围内。

方法声明

```
protected void saveErrors(javax.servlet.http.HttpServletRequest request,
        ActionMessages errors)
```

入口参数

request：HTTP 请求对象。

errors：用于封装错误信息的 ActionMessages 实例。

PDF电子书基地 http://dayo1982.400gb.com

示例

本示例创建 ActionMessage 对象，并通过 saveErrors()方法将其保存在 request 对象中。

```
ActionMessage message = new ActionMessage("user.null");
errors.add("userName", message);
saveErrors(request, errors);
```

13.1.9 saveMessages 方法——保存 ActionMessages 对象

保存 ActionMessages 对象时，如果 request 对象中已经有一个 ActionMessages 对象，则用新的覆盖原有的 ActionMessages 对象。

方法声明

```
protected void saveMessages(javax.servlet.http.HttpServletRequest request,
                            ActionMessages errors)
```

入口参数

request：HTTP 请求对象。

errors：用于封装错误信息的 ActionMessages 实例。

示例

本示例创建 ActionMessage 对象，并通过 saveMessages()方法将其保存在 request 对象中。

```
ActionMessage message = new ActionMessage("user.null");
errors.add("userName", message);
saveMessages(request, errors);
```

13.2 ActionServlet 类——核心控制器组件

org.apache.struts.action.ActionServlet 类是 Struts 框架的核心控制器组件。所有的用户请求都先由 ActionServlet 来处理，再将请求转发给其他的组件。ActionServlet 负责在接收到客户端请求后，将其分发到相应的 Action Bean 处理，再根据处理的结果将不同的显示请求重定向到相应的 JSP 页面中进行显示。

13.2.1 doGet 方法——处理用户 HTTP GET 请求

该方法用于处理用户 HTTP GET 请求。

方法声明

```
protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws java.io.IOException, javax.servlet.ServletException
```

入口参数

request：HTTP 请求对象。

response：HTTP 相应对象。

抛出异常：IOException、ServletException。

示例

本示例在 doGet()方法中设置请求、相应的编码格式，并获取表单数据，在控制台上输出。

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        request.setCharacterEncoding("UTF-8");
        response.setCharacterEncoding("UTF-8");
        String userName = request.getParameter("userName");
                        //获取表单数据
        System.out.println(userName + "登录系统");
    }
```

13.2.2 doPost 方法——处理用户 HTTP POST 请求

该方法用于处理用户 HTTP POST 请求。

方法声明

```
protected void doPost(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
 throws java.io.IOException, javax.servlet.ServletException
```

入口参数

request：HTTP 请求对象。

response：HTTP 相应对象。

抛出异常：IOException、ServletException。

示例

在 doPost()方法中，直接调用 doGet()方法对 post 请求进行处理，这样，保证不管是 get 请求还是 post 请求，处理结果都是一样的。

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        super.doPost(request, response);
        doGet(request, response);
    }
```

13.2.3 destroy 方法——移除控制器组件

移除控制器组件，并释放所占用的资源。

方法声明

```
public void destroy();
```


PDF电子书基地 http://dayo1982.400gb.com

示例

当调用 ActionServlet 实例的 destroy()方法后，会释放 ActionServlet 所占用的资源。本实例在 desctroy()方法中给出提示信息“控制器失效”。

```
public void destroy() {
    System.out.println("该控制器失效");
}
```

13.2.4 init 方法——初始化

ActionServlet 类继承了 javax.Servlet.http.HttpServlet 类，并实现 init()、doGet()、doPost()、destroy()方法。init()方法，Servlet 容器在启动时，或者用户首次请求 ActionServlet 时加载 ActionServlet 类。在 ActionServlet 被加载后立即执行 init()方法。保证 ActionServlet 处理用户请求时已经被初始化。

方法声明

```
protected void init() throws javax.servlet.ServletException
```

抛出异常：ServletException。

示例

本示例在 ActionServlet 子类的 init()方法中，调用 getInitParameter()方法来获得初始化参数 greeting 的值。

```
private String greeting;
public void init() throws ServletException {
    greeting = getInitParameter("greeting");
}
```

13.2.5 initInternal 方法——初始化 Struts 的消息资源

该方法初始化 Struts 框架中内在的消息资源，如与系统日志相关的通知、警告和错误信息。

方法声明

```
protected void initInternal() throws javax.servlet.ServletException
```

抛出异常：ServletException。

示例

本示例在 ActionServlet 子类的 initInternal()方法中，对 MessageResources 实例进行实例化。

```
protected MessageResources interName;
protected void initInternal() throws ServletException {
    String internal = "org.apache.struts.action.ActionResources";
    interName = MessageResources.getMessageResources(internal);
}
```

13.2.6 initOther 方法——加载 ActionServlet 初始化参数

该方法从 web.xml 文件中加载 ActionServlet 的初始化参数，如 config、debug 参数。

方法声明

```
protected void initOther() throws javax.servlet.ServletException
```

抛出异常：ServletException。

示例

本示例在 ActionServlet 子类的 initOther()方法中实现加载 web.xml 中 config 属性。

```
protected String config = "/WEB-INF/struts-config.xml";
protected void initOther() throws ServletException {
    String value;
    value = getServletConfig().getInitParameter("config");
    if (value != null) {
        config = null;
    }
}
```

13.2.7 process 方法——处理接受的请求并做出反应

process()方法处理接受的请求并做出反应。

方法声明

```
protected void process(javax.servlet.http.HttpServletRequest request,
                       javax.servlet.http.HttpServletResponse response)
                throws java.io.IOException, javax.servlet.ServletException
```

入口参数

request：HTTP 请求对象。

response：HTTP 相应对象。

抛出异常：IOException、ServletException。

示例

本实例是 ActionServlet 类的 process()方法的源代码。

```
protected void process(HttpServletRequest request,HttpServletResponse response)throws
IOException,ServletException(
    //根据请求选择恰当的应用模块
    ModuleUtils.getInstance().selectModule(request,getServletContext());
    //获得模块相关的 RequestProcessor 实例
    GetRequestProcessor(getModuleConfig(request)).process(request,response);
    )
```

在上述代码中，虽然 process()方法的实现代码只有几行，但是它所调用的其

PDF电子书基地 http://dayo1982.400gb.com

他方法实现比较复杂。

首先，org.apache.struts.util.RequestUtils 类的静态方法 selectModule()被调用，并将当前请求和 Web 应用程序的上下文环境 ServletContext 作为参数传递到该方法中。selectModule()方法的职责是选择一个应用程序模块来处理当前请求，应用程序模块的选择是通过匹配请求对象的 Servlet 请求路径，即 request 对象的 getServletPath()方法的结果，与每个已配置的应用程序模块的前缀实现的。当只有单个 Struts 配置文件，意味着只有一个原因，即默认应用模块。任何不包含应用模块前缀的请求都将默认应用模块处理。另外，selectModuleConfig()方法将适当的 ApplicationConfig 和 MessageResources 对象存储到请求中，这样其他组件可以很方便地从请求中读取这些对象，从而获得应用配置信息和消息资源。

其次，getModuleConfig()方法取得当前应用模块的配置，getProcessorForModule()方法取得当前应用模块的请求处理器，即 org,apache.struts.action.RequestProcessor 类的实例。如果实例不存在，就创建一个。最终调用 RequestProcessor 类的 process()方法处理了用户的请求。

13.3 DispatchAction 类——请求分发

org.apache.struts.actions.DispatchAction 类继承了 Action 类，该类是一个抽象类，实现了在一个 Action 中完成一组紧密相关的业务操作。创建 DispatchAction 类子类不用覆盖 execute()方法，而是创建与 execute()方法具有同样参数与返回类型的方法。

示例

本实例实现创建 DispatchAction 类的子类 MyAction。

```
public class MyAction extends DispatchAction {
                ……
}
```

13.3.1 execute 方法——处理请求

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

方法声明

```
public ActionForward execute(ActionMapping mapping,ActionForm form,
                javax.servlet.http.HttpServletRequest request,
                javax.servlet.http.HttpServletResponse response)
                throws java.lang.Exception
```

入口参数

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。

抛出异常：Exception。

示例

在 DispatchAction 子类的 execute()方法中，实现当页面跳转。

```
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        return mapping.findForward("welcome");
    }
```

13.3.2 getParameter 方法——获取 parameter 参数值

该方法获取在配置 DispatchAction 子类的 parameter 参数值。

方法声明

```
    protected String getParameter(ActionMapping mapping,ActionForm form,
                    javax.servlet.http.HttpServletRequest request,
                    javax.servlet.http.HttpServletResponse response)
                    throws java.lang.Exception
```

入口参数

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 struts-config.xml 文件中配置的 DispatchAction 子类的 parameter 参数值。

抛出异常：Exception。

示例

在 DispatchAction 子类的自定义方法 add()方法中，调用 getParameter()方法判断提交控制器的请求参数，并转发至相应页面。

```
    public ActionForward add(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        if (getParameter(mapping, form, request, response).equals("method")) {
            return mapping.findForward("error");
        } else {
            return mapping.findForward("welcome");
        }
    }
```


PDF电子书基地 http://dayo1982.400gb.com

13.3.3 getMethodName 方法——返回方法名

通过给定 struts-config.xml 文件中配置的 parameter 参数值，返回 DispatchAction 子类的方法名。

方法声明

```
protected java.lang.String getMethodName(ActionMapping mapping,
ActionForm form, javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response,java.lang.String parameter)
throws java.lang.Exception
```

入口参数

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

prameter：struts-config.xml 文件中配置的 parameter 参数值。

返回值：返回调用控制器组件的方法名。

抛出异常：Exception。

示例

在 DispatchAction 子类的 execute 方法中，实现当用户请求 Action 配置参数是 method，并且调用方法参数为 add 时，转发到指定页面，否则转发至另一个页面。

```
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        if (getMethodName(mapping, form, request, response,
                                "method").equals("add")) {
            return mapping.findForward("error");
        } else {
            return mapping.findForward("welcome");
        }
    }
```

13.4 ForwardAction 类——转发请求

org.apache.struts.actions.ForwardAction 类专门用于转发请求，不执行任何其他业务操作。

示例

本示例创建类 MyAction，该类继承自 ForwardAction 类。

```
public class MyAction extends   ForwardAction {
                    ……
}
```

execute 方法——请求处理

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

方法声明

```
public ActionForward execute(ActionMapping mapping,ActionForm form,
                javax.servlet.http.HttpServletRequest request,
                javax.servlet.http.HttpServletResponse response)
                throws java.lang.Exception
```

入口参数

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。

抛出异常：Exception。

示例

在 ForwardAction 子类的 execute()方法中，实现页面跳转。

```
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        return mapping.findForward("error");
    }
```

13.5 IncludeAction 类——包含其他 Web 组件

org.apache.struts.actions 包中的 IncludeAction 类的意义与 ForwardAction 类似，提供了包含其他 Web 组件的功能。与 ForwardAction 一样，Web 组件通过 IncludeAction 类包含另一个 Web 组件，可以充分利用 Struts 控制器的预处理功能。

示例

本示例创建类 MyAction，该类继承自 InclueAction 类。

```
public class MyAction extends InclueAction{
        ......
}
```

execute 方法——请求处理

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public ActionForward execute(ActionMapping mapping,ActionForm form,
                javax.servlet.http.HttpServletRequest request,
                javax.servlet.http.HttpServletResponse response)
                throws java.lang.Exception
```

入口参数：

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。

抛出异常：Exception。

示例

在 IncludeAction 子类的 execute()方法中，实现页面跳转。

```
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        return mapping.findForward("ok");
    }
```

13.6 LookupDispatchAction 类——多业务处理

org.apache.struts.stions 包中的 LookupDispatchAction 类是 DispatchAction 类的子类，LookupDispatchAction 类也可以定义多业务方法。LookupDispatch Action 类通常被用于在一个表单中有多个提交按钮，并且这些提交按钮有一个共同的名字。

示例

本示例创建类 MyAction，该类继承自 LookupDispatchAction 类。

```
public class MyAction extends LookupDispatchAction{
                ......
}
```

13.6.1 execute 方法——请求处理

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

方法声明

```
public ActionForward execute(ActionMapping mapping,ActionForm form,
                javax.servlet.http.HttpServletRequest request,
                javax.servlet.http.HttpServletResponse response)
                throws java.lang.Exception
```

入口参数：

mapping：用于描述控制器的请求的 ActionMapping 实例。
form：用于保存用户请求表单中的数据的 ActionForm 实例。
request：HTTP 请求对象。
response：HTTP 相应对象。
返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。
抛出异常：Exception。

示例

在 LookupDispatchAction 子类的 execute()方法中，实现页面跳转。

```
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        return mapping.findForward("ok");
}
```

13.6.2 getKeyMethodMap 方法——返回 Map 集合对象

该方法返回包含 key/value 数据的 Map 集合对象。

方法声明

```
protected abstract java.util.Map getKeyMethodMap()
```

返回值：包含 key/valus 数据的 Map 集合对象。Key 和<bean:message>标签的 key 属性匹配，value 和 LookupDispatchAction 子类中的方法匹配。

示例

在 LookupDispatchAction 子类的 getKeyMethodMap()方法中，将<bean:message>标签的 key 标记作为 Map 集合中的 key 标记，将 LookupDispatchAction 子类中的方法名称，作为 Map 集合中的 value 值，并将 Map 集合返回。

```
protected Map getKeyMethodMap() {
        Map map = new HashMap();
        map.put("button.delete.selected", "selected");
        map.put("button.delete.all", "all");
        return map;
    }
```

13.6.3 getMethodName 方法——获取方法名

该方法用于获取方法名。

方法声明

```
protected java.lang.String getMethodName(ActionMapping mapping,
                                         ActionForm form,
javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response,
    java.lang.String parameter) throws java.lang.Exception
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

parameter：与 struts-config.xml 配置文件中<ation>元素中 parameter 属性值匹配。

返回值：返回方法名。

抛出异常：Exception。

示例

在 LookupDispatchAction 子类中，通过 getMethodName()方法，获取提交表单调用的方法名。

```
public ActionForward all(ActionMapping mapping, ActionForm form,
   HttpServletRequest request, HttpServletResponse response) throws Exception {
      String strName = getMethodName(mapping, form,
                                          request, response, "action");
      System.out.println("STRNAME " + strName);
      return mapping.findForward("error");
}
```

13.7 ModuleUtils 类——处理子应用模块

org.apache.struts.util 包中的 ModuleUtils 类，提供了处理子应用模块的实用方法。创建 ModuleUtils 实例，要通过 ModuleUtils 类的 getInstance()方法。

13.7.1 getInstance()方法——返回 ModuleUtils 实例

该方法返回一个 ModuleUtils 实例。

方法声明

```
public static ModuleUtils getInstance()
```

返回值：返回 ModuleUtils 实例。

示例

本示例通过 getInstance()方法获取 ModuleUtils 实例。

```
ModuleUtils mui = ModuleUtils.getInstance()
```

13.7.2 getModuleConfig()方法——返回 ModuleConfig 对象

该方法可在 request 范围内检索并返回 ModuleConfig 对象。

方法声明

```
public ModuleConfig getModuleConfig
(javax.servlet.http.HttpServletRequest request)
```

入口参数：request 是 HTTP 请求对象。

返回值：返回 ModuleConfig 对象。

示例

本示例通过调用 ModuleUtil 类的 getModuleConfig()方法获取 ModuleConfig 对象。

```
ModuleUtils mui = ModuleUtils.getInstance();
ModuleConfig mcg = mui.getModuleConfig(request);
```

13.8 SwitchAction 类——处理多应用模块

org.apache.struts.actions 包中的 SwitchAction 类支持多应用模块，SwitchAction 类用于子应用模块之间的切换。在应用程序模块之间切换时，需要替换不同的应用程序前缀、应用模块的配置 ModuleCofig 以及消息资源，通过 SwitchAction 的实例自动执行了这些操作，简化了应用程序模块之间的切换。

execute 方法——业务请求处理

该方法完成用户请求的业务逻辑，然后根据执行结果把请求转发给其他 Web 组件。

方法声明

```
public ActionForward execute(ActionMapping mapping,ActionForm form,
                javax.servlet.http.HttpServletRequest request,
                javax.servlet.http.HttpServletResponse response)
                throws java.lang.Exception
```

入口参数：

mapping：用于描述控制器的请求的 ActionMapping 实例。

form：用于保存用户请求表单中的数据的 ActionForm 实例。

request：HTTP 请求对象。

response：HTTP 相应对象。

返回值：返回 ActionForward 对象，用于标识要转发到的目标页面。

抛出异常：Exception。

示例

在 SwitchAction 子类的 execute()方法中，实现页面跳转。

```
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
        return mapping.findForward("ok");
    }
```


PDF电子书基地 http://dayo1982.400gb.com

第 14 章　Struts Bean 标签库

14.1　<bean:define>标签——定义一个变量

<bean:define>标签用于定义一个变量。该标签必须存在 id 属性。它指定了一个变量的名称，该变量将成为指定对象的引用。默认情况下该变量被存储在了页面范围内，可以通过 toScope 属性来指定该变量的存储范围。

<bean:define>标签用来定义变量有以下 4 种方式。

语法 1

使用 name 属性指定的一个引用对象来定义变量。该引用对象必须存在于 JSP 的某个范围内，具体语法如下：

```
<bean:define id="var" name=”obj”>
</bean:define>
```

语法 2

同时使用 name 属性和 property 属性指定一个对象来定义变量，具体语法如下：

```
<bean:define id="var" name=”bean” property = “property”>
</bean:define>
```

语法 3

使用 value 属性指定一个字符串复制给 id 属性，具体语法为：

```
<bean:define id="var" value=”value”>
</bean:define>
```

语法 4

将标签体中的内容定义为变量。该方法通过将一个字符串值放置在标签的开始与结束之间给 id 属性定义的变量赋值，具体语法为：

```
<bean:define id=”var”>value</bean:define>
```

<bean:define>标签的各属性表如下。

<bean:define>标签属性表

属　性	类　型	描　述
id	String	用于定义变量的名称
name	String	用于定义一个已经存在的 bean
property	String	该属性用于指定存在的 bean 的某个属性

示例

在 session 范围内定义变量“username_1”，将字符串“Java 编程词典”赋值给该变量。通过<bean:define>标签定义变量“idname”，引用保存在 session 对象中值，并将该值在页面上显示。

```
<% session.setAttribute("username_1","Java 编程词典"); %>
<bean:define id="idname" name="username_1"/>
<bean:write name ="idname"/>
```

在页面中添加如上所示代码后，页面上会显示“Java 编程词典”。

示例

在包 com.ActionForm 下创建类 Bean，该类包含 name 与 passWord 属性，并包含属性对应的 setXXX()与 getXXX()方法。

```
public class Bean {
    private String name = "大熊猫";
    private String passWord = "111";
    public String getPassWord() {
        return passWord;
    }
    public void setPassWord(String passWord) {
        this.passWord = passWord;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

在 jsp 页面中将 com.ActionForm 包中的 Bean 类的 name、passWord 属性值在页面中输出。

```
<jsp:useBean id="userinfo" class="com.ActionForm.Bean"/>
<bean:define id="username_2" name="userinfo" property="name"/>
用户名：<bean:write name="username_2"/>
<br>
<bean:define id="passWord" name="userinfo" property="passWord"/>
密 码:<bean:write name = "passWord"/>
```

本示例运行结果如图 14.1 所示。

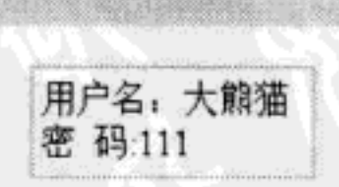


图 14.1 运行结果

示例

定义变量，并通过 value 将变量赋值。

```
<bean:define id="username_2" value="长春"/>
```

示例

定义变量，并通过将字符串值放置在开始与结束之间给 id 属性定义的变量

PDF电子书基地 http://dayo1982.400gb.com

赋值。

```
<bean:define id="like">乒乓球</bean:define>
```

14.2 <bean:cookie>标签——访问 Cookie 对象

<bean:cookie>标签可以访问传入请求中的 Cookie 对象，并将一个指定的 Cookie 对象定义为 JSP 页面变量。Cookie 是用户在访问服务器时，服务器存储在客户端的信息。每个 Cookie 对象可通过其构造方法 Cookie(name,value)来创建。

语法

```
<bean:cookie id="var" name = "cookie1" value = "defvalue" multiple = "str">
</bean:cookie>
```

<bean:cookie>标签的属性表如下。

属　性	类　型	描　述
id	String	指定一个存储 javax.servlet.http.Cookie 类对象的变量
name	String	指定一个 Cookie 对象的名称
value	String	指定一个 Cookie 对象的值
multiple	String	将该属性值设置成任意字符串后，此时 id 属性定义的是 Cookie 类型的数组变量，用来存储具有同一个名称的多个 Cookie 对象

示例

本示例将<bean:cookie>标签定义的 Cookie 对象的 name 属性、value 属性在页面中输出。

```
<bean:cookie id="idcookie" name="username" value="nocookie"/>
Cookie 对象名：<bean:write name="idcookie" property="name"/>
<br>
Cookie 对象值：<bean:write name="idcookie" property="value"/>
```

本示例的运行结果如图 14.2 所示。

Cookie对象名：username
Cookie对象值：nocookie

图 14.2　运行结果

14.3 <bean:header>标签——访问 HTTP 头信息

<bean:header>标签可以访问 HTTP 请求中的标头信息，并将指定的 HTTP 标头信息定义为一个 String 类型的页面变量。

语法

```
<bean:header id="var" name = "cookie1" value = "defvalue" multiple = "str">
</bean:header>
```

<bean:header>标签的属性表如下。

属　性	类　型	描　述
id	String	用于存储指定的 HTTP 标头信息
name	String	用来指定一个 HTTP 标头的名称
value	String	该属性值赋值给 id 属性指定的变量
multiple	String	将该属性值设置成任意字符串后，此时 id 属性定义的是 String 类型的数组变量，用来存储具有同一个名称的多个 HTTP 标头信息。如果请求中具有多个由 name 属性指定的 HTTP 标头名称并且此时忽略了 multiple 属性，则将存储第一个匹配的 HTTP 标头信息

示例

本示例使用<bean:header>标签检索 HTTP 请求 Header 中的“accept-language”信息，将其赋值给 lang 变量。通过<bean:write>标签输出 lang 变量的值。

```
<bean:header id="lang" name="accept-language" value="None"/>
浏览器使用语言：<bean:write name="lang"/>
```

14.4 <bean:parameter>标签——参数设为变量

<bean:parameter>标签将来自 HTTP 请求中指定的参数定义为 JSP 页面变量。

语法

```
<bean:parameter id=”var” name = “cookie1” value = “defvalue”
        multiple = “str”>
</bean:parameter>
```

<bean:parameter>标签的属性表如下。

属　性	类　型	描　述
id	String	定义存储 HTTP 请求中指定参数的值
name	String	指定 HTTP 请求中的参数名
value	String	指定参数的默认值
multiple	String	该属性值设为任意字符串后，此时 id 属性定义的是 String 类型的数组变量，用来存储具有同一个名称的多个参数的值

示例

本示例使用<bean:parameter>标签定义名为“arg”的字符串类型的变量，并设置默认值为“noarg”。通过<bean:write>标签将<bean:parameter>的属性值在页面上输出。

```
<bean:parameter id="arg" name="testrg" value="noarg"/>
<bean:write name="arg"/>
```

14.5 <bean:page>标签——隐含对象设为变量

<bean:page>标签将 JSP 中的某个隐含对象定义为一个页面变量。JSP 隐含对象包括：application、config、session、request 和 response。

PDF电子书基地 http://dayo1982.400gb.com

语法

```
<bean:page id=”var” property =”request”>
</bean:page>
```

<bean:page>标签的属性的属性表如下。

属　性	类　型	描　述
id	String	指定一个存储 JSP 隐含对象的变量
property	String	指定一个 JSP 隐含对象，可选值包括 application、config、request、response 和 session

示例

本示例将 session 对象的有效活动时间在页面上输出。

```
<bean:page id="x_session" property="session"/>
session 有效活动时间：
<bean:write name="x_session" property="maxInactiveInterval"/>
```

运行程序，页面上出现以秒为单位的 session 的有效范围，如图 14.3 所示。

session有效活动时间： 1800

图 14.3　运行结果

14.6　<bean:size>标签——获取集合或数组长度

<bean:size>标签用于获取 Map、Collection 或数组的长度。

语法

```
<bean:size id=”var” collection =”<%=expression%>” name = “cname”
        property=”amethod” scope = “request”>
</bean:size>
```

<bean:size>标签的属性表如下。

属　性	类　型	描　述
id	String	指定一个存储的 Integer 类型变量
collection	String	指定将被计算大小的数组、基于 Map 对象。该属性必须为运行时刻表达式(<%= expression %>)。
name	String	指定某个范围中的存储了将被计算大小的数组、基于 Collection 或 Map 的对象的引用名称
property	String	该属性值匹配由 name 属性指定的对象中的某个 getXXX()方法
scope	String	指定查找由 name 属性指定的对象的范围

示例

本示例在 Jsp 页面中定义 String 数组，并将数组的长度在页面中输出。

```
<%
String[ ] myselect={"A","B","C"};
session.setAttribute("select",myselect);
%>
<bean:size id="idsize_1" collection="<%=myselect%>"/>
数组大小为：<bean:write name="idsize_1"/>
```

14.7 <bean:message>标签——输出资源信息

<bean:message>标签用于输出信息资源文件中文本。在应用程序范围内如果存在多个资源文件，若没有具体的指定，默认情况下会使用名为“Application Resources.properties”的资源文件。在使用<bean:message>标签时可以通过 bundle 属性来指定一个资源文件，其值要与 struts-config.xml 文件中<message-resources>元素的 key 属性值相对应。

语法

```
<bean:message bundle=”var” key =”keyval” name = “bean”
              property = “Jproperty”>
</bean:message>
```

<bean:message>标签的各属性详细介绍如下表所示。

属　性	类　型	描　述
bundle	String	指定资源文件，与 Struts 配置文件中的<message-resources>元素的 key 属性匹配
key	String	指定消息资源文件中一个消息关键字
name	String	指定一个 JavaBean
property	String	指定 JavaBean 的一个属性

示例

在资源文件中定义 key 值为“button.delete.selected”，属性值为“button.delete.ok”的信息，如下所示。

```
button.delete.selected= button.delete.ok
```

在 jsp 页面中通过<bean:message>标签将资源文件中 key 值为“button.delete.selected”的信息在页面中显示。

```
<b/>
<bean:message key="button.delete.selected" />
```

示例

本示例将<bean:message>标签的 name 属性，指定一个可以转化为字符串的变量，该变量的字符串值为信息资源文件中的信息的 key。

```
<%
request.setAttribute("name", "button.delete.selected");
%>
<bean:message name ="name" />
```


PDF电子书基地 http://dayo1982.400gb.com

本示例的运行结果如图 14.4 所示。

button.delete.ok

图 14.4 运行结果

示例

创建类 Bean，该类中包含一个 name 属性，并包含属性的 setXXX()与 getXXX()方法。

```
public class Bean {
    private String name = "大熊猫";
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

将资源文件中的 key 值，设置为类 Bean 中的 name 属性的值为资源文件中的信息的 key 值。通过<bean:messsage>标签将信息资源文件中 key 为“button.delete.selected”的信息在页面中显示。

```
<%
Bean bean = new Bean();
bean.setName("button.delete.selected");
request.setAttribute("selected", bean);
%>
<bean:message name ="selected" property="name"/>
```

14.8 <bean:write>标签——输出信息

<bean:write>标签用于在页面上输出某个 Bean 或它的属性的内容。若要输出的对象是 String 类型，则输出该字符串，否则调用该对象的 toString()方法后输出相应的字符串。

语法

```
<bean:write name = ”Name” property = “Property” filter = “true”
            bundle = “Bundle” format = “Format”>
</bean:write>
```

<bean:write>标签的各属性详细介绍如下表所示。

属　性	类　型	描　述
name	String	指定 JSP 范围中存储的某一对象的引用名
property	String	指定 JavaBean 的属性
filter	boolean	是否对 HTML 中的敏感字符进行转换。默认值是“true”，若设置为“false”将不进行转换

续表

属 性	类 型	描 述
bundle	String	用来指定资源文件
format	String	指定一个格式化将要输出指定内容的格式字符串

示例

在 com.ActionForm 包中创建类 Bean，该类中包含属性 name。并包含其 setXXX()与 getXXX()属性。

```
public class Bean {
    private String name = "Java 编程词典";
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

在 jsp 页面中通过<jsp:useBean>标签引用类 Bean，并通过<bean:write>标签将 bean 类中的 name 属性在页面中显示。

```
<jsp:useBean id="bean" class="com.ActionForm.Bean"/>
<bean:write name ="bean" property="name"/>
```

14.9 <bean:resource>标签——检索资源内容

<bean:resource>标签用于检索 Web 资源的内容。

语法

```
<bean:resource id = "var" name = "address" input = "str">
</bean:resource>
```

<bean:resource>标签的各属性详细介绍如下表所示。

属 性	类 型	描 述
id	String	指定一个存储指定资源内容的变量
name	String	指定应用程序资源位置
input	String	如果没有设置 input 属性，则 id 属性的变量为字符串类型；如果给 input 属性设置了值，则 id 属性定义的变量为 java.io.InputStream 类型

示例

如果当前应用根目录下存在 index.jsp 和 errors.jsp 文件。若想在 index.jsp 页面中

PDF电子书基地 http://dayo1982.400gb.com

加载 errors.jsp 文件并输出该文件的源代码，可在 index.jsp 文件中编写如下代码。

```
<bean:resource id="idtop" name="errors.jsp"/>
<pre>top.jsp 源代码：
<bean:write name="idtop" filter="true"/>
```

运行 index.jsp 页面，将 errors.jsp 页面中的源代码在页面中显示，如图 14.5 所示。

```
top.jsp源代码：
<%--
    Document   : errors
    Created on : 2008-11-25, 14:25:43
    Author     : Administrator
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
           <html:form action="/welcome.do">
        <html:submit value="okbutton"/>
        </html:form>
    </body>
</html>
```

图 14.5 运行结果

14.10 <bean:include>标签——包含 Web 资源

该标签类似于<jsp:include>标记，用来包含其他 Web 资源的内容。但<bean:include>标签将指定的资源内容存储到一个页面变量中，并不像<jsp:useBean>标记直接显示到网页中。

语法

```
<bean:include id = "var" forward = "address" page = "/uri" herf = "url">
</bean:include>
```

<bean:include>标签的各属性详细介绍如下表所示。

属　性	类　型	描　述
id	String	指定一个存储指定资源内容的变量
forward	String	指定了匹配 Struts 配置文件中<global-forwards>元素中定义的某个<forward>子元素的 name 属性值的 URI 地址
page	String	指定了相对于当前应用的 URI 地址，以“/”开头
href	String	指定一个完整的 URL 地址

示例

将当前应用的根目录下存在的 index.jsp、top.jsp、end.jsp，通过<bean:include>标签将其包含在一个页面中。

```
<bean:include id="idtop" page="/top.jsp"/>
<bean:write name="idtop" filter="false"/>
<hr>This is Body!<hr>
<bean:include id="idend" page="/end.jsp"/>
<bean:write name="idend" filter="false"/>
```

注意

在使用了该标签将应用中的其他资源包含进当前页面后，可使用<bean:write>标签进行显示。值得注意的是如果被包含的资源是网页文件(如: *.jsp 或*.html)，则在使用<bean:write>标签进行显示时，要将 filter 属性值设为“false”取消过滤功能，否则将输出被包含网页的源代码。

14.11 <bean:struts>标签——获取 Struts 对象

通过该标签可以在页面中对 Struts 中的 3 个重要的配置对象进行引用。这 3 个对象分别是 FormBeanConfig、ForwardConfig 和 ActionConfig。

语法

```
<bean:struts id = “var” formBean = “beanName” forward = “forwards”
             mapping = “actionMapping”>
</bean:struts>
```

<bean:struts>标签的各属性详细介绍如下表所示。

属　性	类　型	描　述
id	String	指定一个存储 Struts 配置对象的变量
formBean	String	指定一个 ActionForm Bean 对象。与当前应用中的 struts-config.xml 配置文件中<form-beans>元素的 <form-bean>子元素中的 name 属性值匹配
forward	String	指定一个 AcitonForward 对象。与当前应用中 struts-config.xml 配置文件中<global-forwards>元素的 <forward>子元素中的 name 属性值匹配
mapping	String	指定一个 ActionMapping 对象。当前应用中 struts-config.xml 文件中<action-mappings>元素的 <action>子元素中的 path 属性值匹配

示例

本示例引用一个名为“MyForm”的 formBean 对象，通过<bean:write>标签将 formBean 的 name 属性在页面中显示。

PDF电子书基地 http://dayo1982.400gb.com

```
<bean:struts id="idaction" formBean="MyForm"/>
传递的参数为：<bean:write name="idaction" property="name"/>
```

在 struts 配置文件中与之匹配的<form-bean>元素的配置信息为：

```
<form-bean name = "MyForm" type="com.ActionForm.MyForm"></form-bean>
```

示例

本示例将 ActionMapping 地址是“welcome”的 Action 对象的 parameter 属性在页面中输出。

```
<bean:struts id="idaction" mapping="/welcome"/>
传递的参数为：<bean:write name="idaction" property="parameter"/>
```

在 struts 配置文件中与之匹配的<action>元素的配置信息为：

```
<action path="/welcome" name="MyForm" type = "com.myAction.struts.MyAction" input =
"/index.jsp" parameter="action"/>
```

第 15 章　Struts HTML 标签库

Struts HTML 标签可以和标准的 HTML 元素完成相等的功能，并且这些标签可以和 Struts 框架的其他组件紧密地联系在一起。

15.1　<html:base>标签——获得页面 UTL

该标签对应 HTML 中的<base>元素。<html:base>在网页中的<head>部分生成<base>元素。

语法

```
<html:base server = "serverName" target = "blank"/>
```

<html:base>标签属性如下表所示。

属　性	类　型	描　述
server	String	设置请求链接中服务器的名称
target	String	指定打开目标页面的窗口或框架

示例

本示例使用<html:base>标签设置请求链接中服务器的名称等信息。

```
<html:html lang="true">
<head>
    <html:base server="192.168.1.128" target="_blank"/>
</head>
<body>
    <a href="default.jsp">链接到 XXX 网站</a>
</body>
</html:html>
```

15.2　<html:checkbox>标签——生成复选框

<html:checkbox>标签用于生成复选框组件。对应着 HTML 中的<input type =”checkbox”>元素。

语法

```
<html:checkbox property = "checkbox1" value = "true">
</html:checkbox>
```

<html:checkbox>标签属性如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

属　性	类　型	描　述
property	String	与表单关联的 ActionForm Bean 中的 getXXX()方法与 setXXX()方法相匹配
value	boolean	检查框选中时添加到请求中的参数值

示例

本示例使用<html:checkbox>标签向表单中添加复选框组件。

```
<html:form action="/welcome.do">
    爱好:<br>
    <html:checkbox property="checkbox1">足球</html:checkbox>
    <html:checkbox property="checkbox1">蓝球</html:checkbox>
    <html:checkbox property="checkbox1">上网</html:checkbox>
    <html:checkbox property="checkbox1">旅游</html:checkbox>
</html:form>
```

运行结果如图 15.1 所示。



图 15.1　运行结果

15.3　<html:errors>标签——输出错误信息

<html:errors>标签用于输出错误信息。<html:errors>标签在 request 或 session 范围内寻找 ActionMessages(或其子类 ActionErrors)集合对象，在 ActionMessages 集合对象中读取 ActionMessage 对象，把 ActionMessage 对象包含的消息文本显示到网页中。

语法

```
<html:errors name = "keyWords" property = "keyName" bundle = "properties"/>
```

<html:errors>标签属性如下表所示。

属　性	类　型	描　述
name	String	指定 ActionMessages 对象存放在 request 或 session 范围内的属性 key
property	String	指定属性信息
bundle	String	指定消息资源文件

注意

如果没有指定 property 属性，将显示 ActionMessages 对象中所有的 ActionMessage；如果没有指定 bundle 属性，将从应用默认的 Resource Bundle 中获取消息文本。

示例

在 jsp 页面中设置“用户名”、“密码”两个文本框，通过<html:errors>标签输出错误信息。

```
<html:form action="/welcome.do"><center>
        用户名：<html:text property="name" size="10"/>
        <font color="red">
            <html:errors property="un"/>
        </font><br>
        密　　码 ：<html:text property="passWord" size="10"/>
        <font color="red">
            <html:errors property="pw"/>
        </font><br>
        <html:submit value="tijiao"/>
    </center>
</html:form>
```

用户提交“welcome.do”后，根据 Struts 中的配置信息，系统将提交指定的 Action。在 Action 的 execute()方法中判断表单提交的“用户名”、“密码”是否为空，如果为空则向 ActionMessages 对象中添加 ActionMessage 对象，关键代码如下：

```
ActionMessages errors = new ActionMessages();
        if (myForm.getName().equals("")) { //判断用户名是否为空
            errors.add("un", new ActionMessage("userName.is.null"));
            saveErrors(request, errors);
            return mapping.findForward("welcome");
        }
        if (myForm.getPassWord().equals("")) {//判断密码是否为空
            errors.add("pw", new ActionMessage("passWord.is.null"));
            saveErrors(request, errors);
            return mapping.findForward("welcome");
        }
```

15.4 <html:file>标签——上传文件

<html:file>标签可实现文件上传功能。

语法

```
<html:file property = “file”/>
```

<html:file>标签属性如下表所示。

属　性	类　型	描　述
property	String	与 ActionForm 中 FormFile 类型的属性对应

PDF电子书基地 http://dayo1982.400gb.com



注意

在使用<html:file>时应注意<html:file>标签要嵌套在<html:form>标签中。<html:form>标签的 method 属性必须设为“POST”，<html:form>标签的编码类型 enctype 属性必须设为“multipart/form-data”。

示例

本示例在 jsp 页面中添加<html:file>标签，实现文件上传。

```
<html:form action="/welcome.do" method="POST"
          enctype="multipart/form-data"><center>
        请选择上传的文件：<html:file property="file"/><br>
        <html:submit value=""/>
    </center>
```

15.5 <html:form>标签——创建表单

<html:form>标签用于创建表单。该标签中必须包含 action 属性，否则会抛出异常。

语法

```
<html:form action = “action” method = “get” enctype= “data”
           focus=“un”>
</html:form>
```

<html:form>标签属性如下表所示。

属　性	类　型	描　述
action	String	用来指定表单提交后请求的组件
method	String	用来指定表单数据传至服务器的方式
enctype	String	用来指定表单的编码格式
focus	String	设置页面被加载时焦点位置

示例

本示例通过<html:form>实现创建表单，并指定表单提交时请求的组件，以及表单数据传至服务器的方式。

```
<html:form action="/welcome.do" method="POST">
            ... ...//其他设计表单代码
</html:html>
```

15.6 <html:hidden>标签——生成隐藏属性

<html:hidden>标签用来在表单中生成隐藏字段，隐藏字段用于在表单上存放不希望让用户看到或不允许修改的信息，并可以在请求中传递该值。

语法

```
<html:hidden property = “var” name = “beanName” write = “false”>
</html:hidden>
```

<html:hidden>标签属性如下表所示。

属　性	类　型	描　述
property	String	用于定义属性名
name	String	指定了填充该元素的 Bean
write	boolean	该属性的默认值为“false”，将该属性值设为“true”时，可以将隐藏的值输出到页面进行显示

示例

本示例在表单中定义隐藏字段，该隐藏字段的 property 属性值为“hidden”，与指定 Form Bean 的“hidden”属性对应。

```
<html:form action="/welcome.do" method="POST">
        <html:hidden property="hidden"/>
            … …//其他设计表单代码
</html:html>
```

15.7 <html:html>标签——<html>元素

<html:html>标签用于在网页的开头生成 HTML 的<html>元素。标志着一个页面的开始和结束。

语法

```
<html:html lang = “true”>
</html:html>
```

<html:html>标签属性如下表所示。

属　性	类　型	描　述
lang	String	用于显示用户使用的语言

当<html:html>标签中的 lang 属性设置为“true”时，<html:html>标签会先在当前会话中查找 Locale 对象，并根据该 Locale 对象输出网页使用的语言。如果不存在 session 会话或者当前会话中没有 Locale 对象，则根据客户端的 HTTP 请求头中的 accept-language 属性值来输出网页使用的语言。若 HTTP 请求头中不存在 accept-language 属性，此时<html:html>标签会根据默认的 Locale 来输出网页使用的语言。

示例

本示例是 jsp 页面中一段代码，使用<html:html>标签定义网页的开始与结尾。

```
<%@ taglib uri="struts-html" prefix="html"%>
<html:html lang="true">
<body>Hello world!</body>
</html:html>
```

15.8 <html:img>标签——嵌入图片

<html:img>标签用于在 HTML 页中嵌入图片。此外，它还允许包含请求变量，

PDF电子书基地 http://dayo1982.400gb.com

以便动态控制图片的输出。

语法 1

```
<html:img src = "URL" height = "height" width = "width" alt = "str">
</html:img>
```

使用 src 属性指定的是一个完整的 URL 地址。引用的地址为：协议+主机+端口+src 的属性值。src 属性值应以“/”开头。

语法 2

```
<html:img page= "URL" height = "height" width = "width" alt = "str">
</html:img>
```

使用 page 属性值应以“/”开头，引用的地址为：当前应用程序路径+page 属性值。

<html:img>标签属性如下表所示。

属　性	类　型	描　述
arc	String	指定引用图片的地址
page	String	指定引用图片文件所在的相对路径
height	int	图像的高度，单位为像素
width	int	图像的宽度，单位为像素
alt	String	如果找不到指定文件或客户端浏览器不支持图片显示时，显示的替代文本

示例

本示例在表单中使用<html:text>标签添加文件框，使用<html:img>标签在页面中显示“imgaes”文件夹下名称为“9.gif”的图片文件。

```
<html:form action="/welcome.do">
    姓名：<html:text property="name" value="张三"/><br>
    性别：<html:text property="sex" value="女"/><br>
    照片：<html:img page="/images/9.gif"/>
</html:form>
```

本示例运行结果如图 15.2 所示。

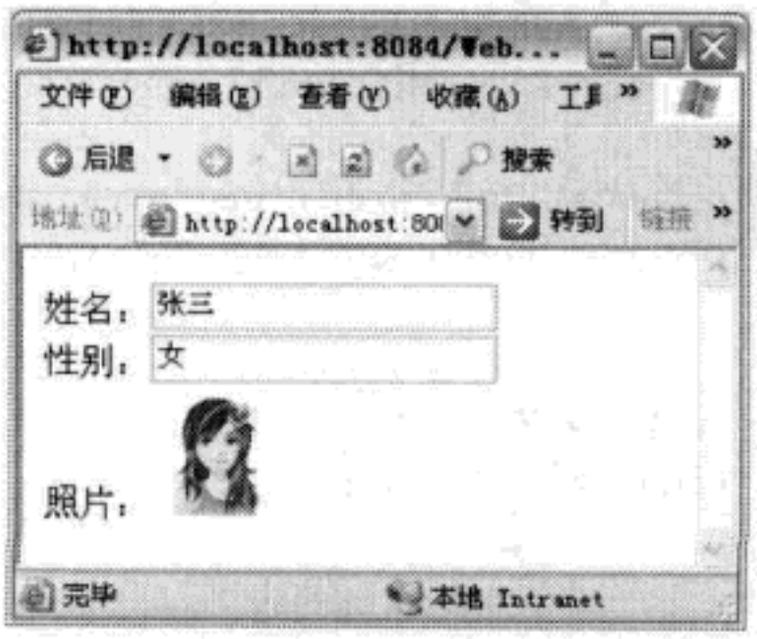


图 15.2　运行结果

15.9 <html:image>标签——生成< img>标签

<html:image>标签与标签类似，都可以在页面中显示图片，并且可以通过单击该标签生成的图片来提交表单。

语法 1

```
<html:image src = "URL" height = "height" width = "width" alt = "str">
</html:image>
```

使用 src 属性指定的是一个完整的 URL 地址。引用的地址为：协议+主机+端口+src 的属性值。src 属性值应以“/”开头。

语法 2

```
<html:image page= "URL" height = "height" width = "width" alt = "str">
</html:image>
```

使用 page 属性值应以“/”开头，引用的地址为：当前应用程序路径+page 属性值。

<html:image>标签属性如下表所示。

属　性	类　型	描　述
arc	String	指定引用图片的地址
page	String	指定引用图片文件所在的相对路径
height	int	图像的高度，单位为像素
width	int	图像的宽度，单位为像素
alt	String	如果找不到指定文件或客户端浏览器不支持图片显示时，显示的替代文本

示例

本示例将“图片一”与“图片二”嵌套在一个表单中，实现单击任意图片后，都会提交表单。

```
<html:form action="/welcome.do">
    <table border="1" width="200">
        <tr bgcolor="lightgrey" height="25">
     <td align="center" colspan="2">使用&lt;html:image&gt;标签！ </td></tr>
        <tr>
            <td align="center"><html:image page="/images/9.gif"/></td>
            <td align="center"><html:image page="/images/5.gif"/></td>
        </tr>
        <tr>
            <td align="center">图片一</td>
            <td align="center">图片二</td>
        </tr>
        <tr><td colspan="2"></td></tr>
    </table>
</html:form>
```


PDF电子书基地 http://dayo1982.400gb.com

本实例运行结果如图 15.3 所示。



图 15.3 运行结果

15.10 <html:link>标签——生成<a>标签

<html:link>标记用于生成一个 HTML 超链接，可以通过单击超链接将浏览器的当前显示定位到被标记的位置。

<html:link>标签创建的超链接有以下 4 种方式。

语法 1

超链接地址指定匹配 Struts 配置文件中<action>元素中的 path 属性值的 URI 地址。

```
<html:link action = "address">
</html:link>
```

语法 2

超链接地址指定匹配 Struts 配置文件中<global-forwards>元素中定义的某个<forward>元素的 name 属性值的 URI 地址。

```
<html:link forward = "address">
</html:link>
```

语法 3

指定完整的 URL。

```
<html:link href = "address">
</html:link>
```

语法 4

指定相对于当前网页的 URL 地址。

```
<html:link page = "address">
</html:link>
```

<html:link>标签属性如下表所示。

属　性	类　型	描　述
action	String	指定超链接的地址，与<action>配置文件中 path 属性值的 URI 地址匹配
forward	String	指定超链接的地址，与<global-forwards>元素中定义的某个<forward>元素的 name 属性值的 URI 地址匹配
href	String	指定超链接地址
page	String	指定超链接地址，以“/”开头

示例

本示例通过<html:link>标签向页面中添加超链接。

```
<html:form action="/welcome.do">
    <html:link action="welcome">
            通过 Action 属性设置超链接</html:link><br>
    <html:link forward="welcome">
            通过 forward 属性设置超链接</html:link><br>
</html:form>
```

15.11 <html:messages>标签——输出信息

<html:messages>标签用于在页面上输出信息。

语法

```
<html:messages id = “var” name = “key” message = “true”>
</html:messages>
```

<html:messages>标签属性如下表所示。

属　性	类　型	描　述
id	String	用来指定一个 JSP 页面变量，与<bean:write>标签的 name 属性匹配
name	String	指定 ActionMessages 对象存放在 request 或 session 范围内的属性的 key
message	boolean	指定信息的来源

示例

在 Action 中通过 saveErrors()或 saveMessages()方法将 ActionMessages 对象存入 request 或 session 范围内。

```
ActionMessage messages=new ActionMessages();
errors.add("username",new ActionMessage("com.logon.name.error"));
saveErrors(request,messages);
```

在 JSP 页面中使用<html:messages>标签输出消息的代码如下。

```
<html:messages id="messageid" message="true">
    <bean:write name="messageid"/>
</html:messages>
```


PDF电子书基地 http://dayo1982.400gb.com

15.12 <html:multibox>标签——生成复选框

<html:multibox>标签可以生成复选框。<html:multibox>可以动态决定被选中的复选框的数目。

语法

```
<html:multibox property = "var">
</html:multibox>
```

<html:multibox>标签属性如下表所示。

属　性	类　型	描　述
property	String	指定属性名称

示例

本示例通过使用<html:multibox>标签完成多项选择。创建 ActionForm Bean，在 Bean 中包含两个 String 类型数组，及其对应的 setXXX()与 getXXX()方法：

```
private String[] allselects = {"A", "B", "C", "D"};
    private String[] myselects = new String[0];
    public void setMyselects(String[] myselects) {
        this.myselects = myselects;
    }
    public String[] getMyselects() {
        return this.myselects;
    }
    public void setAllselects(String[] allselects) {
        this.allselects = allselects;
    }
    public String[] getAllselects() {
        return this.allselects;
    }
```

JSP 页面中完成显示多项选择：

```
<jsp:useBean id="selects" class="com.ActionForm.UserNameForm"/>
<table border="1" >
    <tr><td align="center">数组检查框</td></tr>
    <tr><td align="center"><html:errors property="name"/></td></tr>
    <tr>
<td>多项选择：
    <logic:iterate id="idselect" name="selects" property="allselects">
        <html:multibox property="myselects">
            <bean:write name="idselect"/>
        </html:multibox>
        <bean:write name="idselect"/>
    </logic:iterate>
```

```
            </td>
        </tr>
        <tr><td align="center"><html:submit value="提交"/></td></tr>
</table>
```

本示例运行结果如图 15.4 所示。

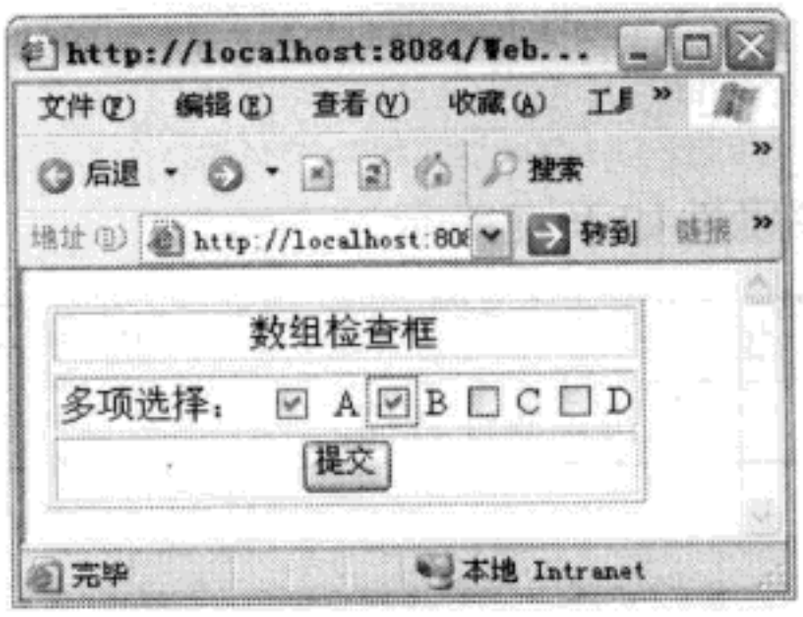


图 15.4 运行结果

15.13 <html:option>标签——生成列表选项

<html:option>标签需要嵌套在<html:option>标签中生成一个下拉列表项。

语法 1

```
<html:option value = “var”>var</html:option>
```

语法 2

```
<html:option value = “var” bundle = “kvar” key = “kname”>var</html:option>
```

<html:option>标签属性如下表所示。

属　性	类　型	描　述
value	String	表示选项的值，该值会通过请求被传递到服务器中
bundle	String	指定了资源文件的引用名
key	String	对应 bundle 属性指定的资源文件中的消息关键字

示例

本示例使用<html:option>标签添加列表项。

```
<html:form action="/welcome.do">
    姓名：<html:text property="username"/>
    性别：<html:select size="1" property="usersex">
        <html:option value="man">男</html:option>
        <html:option value="women">女</html:option>
    </html:select><br>
</html:form>
```


PDF电子书基地 http://dayo1982.400gb.com

本示例运行结果如图 15.5 所示。

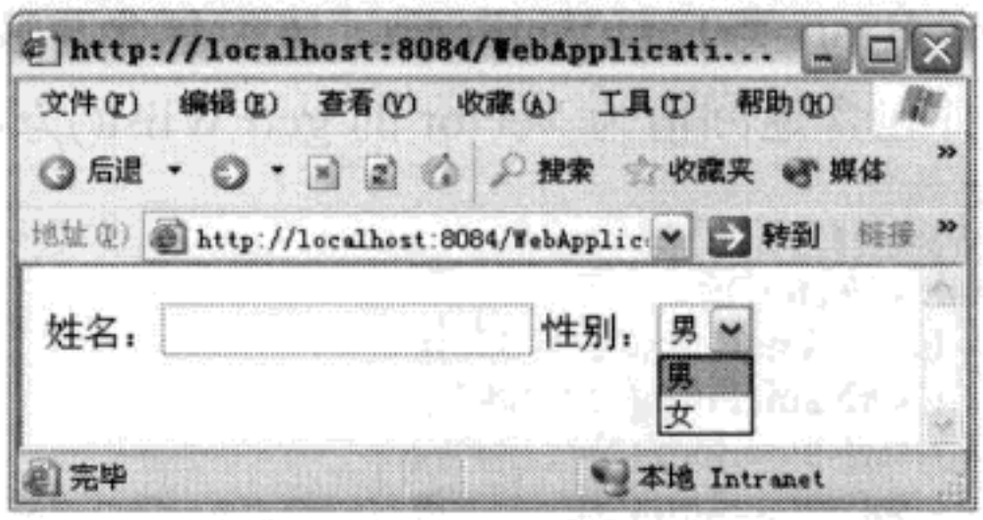


图 15.5 运行结果

示例

本示例使用<html:option>标签，将信息资源文件中 key 值为“user.sex.women”与“user.sex.man”的信息添加到列表中。

```
<html:select property="usersex">
    <html:option value="man" bundle="sex" key="user.sex.man"/>
    <html:option value="woman" bundle="sex" key="user.sex.women"/>
</html:select>
```

15.14 <html:options>标签——生成多个列表

<html:options>标签被嵌套在<html:select>标签中生成一组下拉列表项。该标签提供了一组 HTML<option>元素。

语法

```
<html:options collection = “kname” property="fw_value"
              labelProperty="fw_name">
</html:options>
```

<html:options>标签属性如下表所示。

属　性	类　型	描　述
collection	String	指定的存在某个范围中的集合来生成列表项
property	String	表示下拉列表的实际值
labelProperty	String	表示下拉列表的标签

示例

本示例使用<html:options>标签向下拉列表中添加列表项。首先，创建 ActionForm Bean，该 Bean 中包含两个 String 对象、带有参数的构造方法。关键代码如下：

```
private String fw_name;
private String fw_value;
public UserNameForm(String fw_value, String fw_name) {
    this.fw_name = fw_name;
```

```
        this.fw_value = fw_value;
    }
```

创建 ActionForm Bean 类 FangWeiForm，该类包含与表单属性对应的 String 类型变量 fangwei，并包含返回值为 Vector 的 getFWlist()方法，关键代码为：

```
private String fangwei;
public Vector getFWlist(){
    Vector fwlist=new Vector();
    fwlist.add(new UserNameForm("D","东"));
    fwlist.add(new UserNameForm("X","西"));
    fwlist.add(new UserNameForm("N","南"));
    fwlist.add(new UserNameForm("B","北"));
    return fwlist;
}
```

在 JSP 页面中使用<html:options>标签，将 FangWeiForm 中定义的集合添加到下拉列表中。

```
<jsp:useBean id="fws" class="com.ActionForm.FangWeiForm"/>
<%
session.setAttribute("allfw", fws.getFWlist());
%>
<html:form action="/welcome.do">
    <table border="1" width="250">
        <tr bgcolor="lightgrey" height="20">
            <td align="center">使用自行编写的类创建下拉列表</td></tr>
        <tr><td align="center">
            方位:
            <html:select property="fangwei">
            <html:options collection="allfw" property="fw_value"
                    labelProperty="fw_name"/>
            </html:select>
            <html:submit value="提交"/>
        </td></tr>
    </table>
```

本示例运行结果如图 15.6 所示。

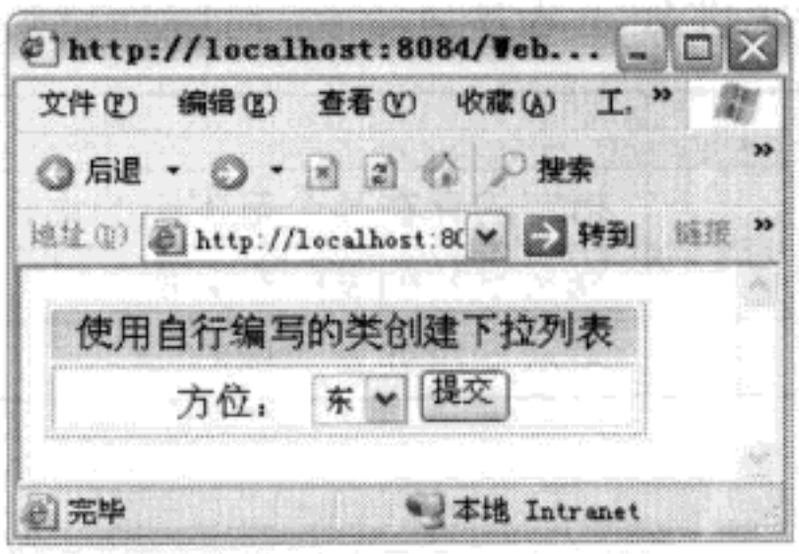


图 15.6 运行结果

15.15 <html:optionCollection>标签——下拉列表

<html:optionCollection>标签与<html:options>标签相似，用于在<html:select>

PDF电子书基地 http://dayo1982.400gb.com

标签中生成一组下拉列表项。

语法

```
<html: optionCollection property = “kname” value="var"
                    label="fname">
</html: optionCollection>
```

<html:optionCollection>标签属性如下表所示。

属　性	类　型	描　述
property	String	指定一个对象，该对象中的每个元素为一个 Bean
value	String	表示下拉列表的实际值
label	String	表示下拉列表的标签

示例

创建类 MyAge 类，该类中包含两个 String 类型的属性，并包含带有参数的构造方法。

```
public class MyAge {
    private String name;
    private String truename;
    public MyAge(String name,String truename){
        this.name=name;
        this.truename=truename;
    }
    ……//省略了属性的 getXXX()与 setXXX()方法
}
```

创建表单对应的 ActionForm Bean 类 AgeForm，关键代码如下：

```
private ArrayList myage = new ArrayList();
    private String age;
    public UserNameForm() {
        myage.add(new MyAge("鼠", "1"));
        myage.add(new MyAge("牛", "2"));
        myage.add(new MyAge("虎", "3"));
    }
```

在 JSP 页面中使用<html:optionsCollection>标签，向下拉列表中添加列表项。

```
<table border="1" width="200">
  <tr bgcolor="lightgrey" height="25">
<td align="center">&lt;html:optionsCollection&gt;标签</td></tr>
  <tr><td align="center">属相：
      <html:select property="age">
    <html:optionsCollection property="myage"
                            value="truename" label="name"/>
          </html:select>
          <html:submit value="提交"/>
  </td></tr>
</table>
```

本示例运行结果如图 15.7 所示。



图 15.7 运行结果

15.16 <html:password>标签——生成密码框

<html:password>标签用于生成一个密码输入的组件。该标签要嵌套在<html:form>标签中使用。

语法

```
<html:password property = “var” name="bname"
                redisplay="true">
</html:password>
```

<html:password>标签属性如下表所示。

属　性	类　型	描　述
property	String	指定标签名称
name	String	指定了填充该元素的 Bean
redisplay	boolean	将该属性值设置为“false”时不会重新显示上次在表单中输入的内容，默认值为“true”

示例

本示例使用<html:text>标签生成文本框组件，使用<html:password>标签生成密码组件。

```
用户名：<html:text property="name"/>
密　码：<html:password property="password" redisplay="false"/>
```

15.17 <html:radio>标签——生成单选按钮

<html:radio>标签用来生成单选按钮。该标签应嵌套在<html:form>标签中使用。

语法

```
<html: radio property = “var” name="bname" value="pvalue">
</html: radio>
```


PDF电子书基地 http://dayo1982.400gb.com

<html:radio>标签属性如下表所示。

属　性	类　型	描　述
property	String	指定标签名称
name	String	指定存在某一范围内的 Bean
value	String	表示该控件的实际值

示例

本示例使用<html:radio>标签在页面中添加单选按钮。

```
<table border="1" width="200">
    <tr bgcolor="lightgrey" height="25"><td align="center">&lt;html:radio&gt;标签</t d> </tr>
    <tr><td align="center">性别：
            <html:radio property="sex" value="man"/>男
            <html:radio property="sex" value="women"/>女
    </td></tr>
    <tr><td align="center"><html:submit value="提交"/></td></tr>
</table>
```

本实例运行结果如图 15.8 所示。

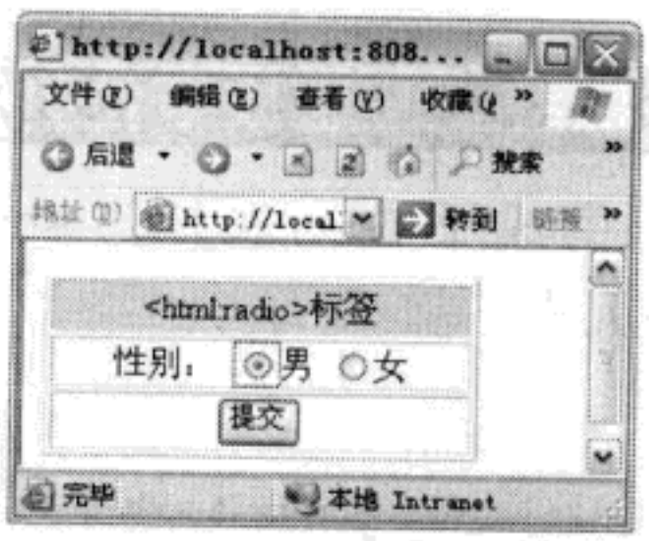


图 15.8　运行结果

15.18　<html:submit>标签——生成提交按钮

<html:submit>标签生成表单的提交按钮，该标签要嵌入到<html:form>标签中。

语法

```
<html:submit value = “var”>
</html:submit>
```

<html:submit>标签属性如下表所示。

属　性	类　型	描　述
value	String	表示该控件的实际值

示例

本示例在页面中添加提交按钮，单击提交按钮后，提交表单。

```
<table border="1" width="200">
    <tr bgcolor="lightgrey" height="25"><td align="center">
        &lt;html:submit&gt;标签</td></tr>
    <tr><td align="center">进入主页面：
    </td></tr>
    <tr><td align="center"><html:submit value="提交"/></td></tr>
</table>
```

本实例运行结果如图 15.9 所示。



图 15.9　运行结果

15.19　<html:select>标签——生成下拉列表框

<html:select>标签用于生成下拉列表框。<html:select>标签要嵌入到<html:form>标签后生成表单元素。

语法

```
<html:select property = "var" size = "size" multiple = "yes">
</html:select>
```

<html:select>标签属性如下表所示。

属　性	类　型	描　述
property	String	表示组件的名称
size	int	指定显示在页面中可见的列表项数目
multiple	String	表示是否支持同时选择多项

示例

本示例在实现创建下拉列表，并通过<html:select>标签向列表中添加列表项。

```
所在地区：
<html:select property="shengfen" size="1">
  <html:option value="JL">吉林</html:option>
  <html:option value="SH">上海</html:option>
  <html:option value="SZ">深圳</html:option>
</html:select>
```


PDF电子书基地 http://dayo1982.400gb.com

本示例运行结果如图 15.10 所示。



图 15.10 运行结果

15.20 <html:text>标签——生成文本框

<html:text>标签用于在 HTML 页面中生成单行文本输入组件，该标签被嵌套在<html:form>标签中成为表单元素。

语法

```
<html:text property = "var" name = "bname"/>
```

<html:text>标签属性如下表所示。

属　性	类　型	描　述
property	String	表示组件的名称
name	String	指定填充该元素的 Bean

注意

property 属性与 ActionForm Bean 中的一个属性匹配。

示例

本示例使用<html:text>标签创建 3 个文本框。

```
姓名：<html:text property="name"/><br>
年龄：<html:text property="password"/><br>
性别：<html:text property="sex"/><br>
```

15.21 <html:textarea>标签——生成文本域

<html:textarea>标签用于生成多行多列的文本输入组件。该标签被嵌套在<html:form>标签中成为表单的元素。

语法

```
<html:textarea property = "var" name = "bname" rows = "size" cols = "size"/>
```

<html:textarea>标签属性如下表所示。

属　性	类　型	描　述
property	String	表示组件的名称
name	String	指定填充该元素的 Bean
rows	int	指定要显示的行数
cols	int	指定要显示的列数

注意

property 属性与 ActionForm Bean 中的一个属性匹配。

示例

本示例实现在表单中创建文本域，并指定文本域中可输入的文本的行数与列数。

```
<html:form action="/welcome.do">
描述：<html:textarea property="context" cols="10" rows="10"/>
</html:form>
```


PDF电子书基地 http://dayo1982.400gb.com

第16章　Struts logic 标签库

Struts Logic 标签库中的标签可以根据特定的逻辑条件来控制输出页面的内容，或者循环遍历集合中的所有元素。

16.1　<logic:equal>标签——比较对象是否相等

<logic:equal>标签用来比较变量是否等于指定常量。如果条件成立，Struts会执行开始标签与结束标签之间的代码。

语法

```
<logic:equal vlaue = "value" cookie = "cname" header = "head" name = "bname"
             parameter = "var" property = "bean" scope = "request">
</logic:equal>
```

<logic:equal>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	.String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例通过 header 属性指定的 HTTP 请求中的标头信息作为比较数值，若客户端浏览器使用的是中文，则将字符串“当前浏览器使用的是中文语言!”输出。

```
<logic:equal value="zh-cn" header="accept-language">
    当前浏览器使用的是中文语言！ <br>
</logic:equal>
```

16.2　<logic:notEqual>标签——比较对象是否不相等

<logic:notEqual>标签比较数值是否不等。如果条件成立，Struts 会执行开始标签与结束标签之间的代码。

语法

```
<logic:notEqual vlaue = “value” cookie = “cname” header = “head”
    name = “bname” parameter = “var” property = “bean” scope = “request”>
</logic:notEqual>
```

<logic:notEqual>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例首次创建类 MyAge 对象，并设置其 name 属性。通过<logic:notEqual>判断其 name 属性是否不等于“tom”。

```
<%
MyAge myage = new MyAge();
myage.setName("Linda");
request.setAttribute("beanName", myage);
%>
<logic:notEqual name="beanName" property="name" value="Tom">
    The name of beanName is not "Tom" <br>
</logic:notEqual>
```

16.3 <logic:greaterEqual>标签——比较是否大于或等于

<logic:greaterEqual>标签比较变量是否大于或等于指定的常量。如果条件成立，Struts 会执行开始标签与结束标签之间的代码。

语法

```
<logic:greaterEqual vlaue = “value” cookie = “cname” header = “head”
    name = “bname” parameter = “var” property = “bean” scope = “request”>
</logic:greaterEqual>
```

<logic:greaterEqual>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名

PDF电子书基地 http://dayo1982.400gb.com

续表

属　性	类　型	描　述
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例首先通过 session 对象的 setAttribute()方法将整型数“123”保存在 Session 范围内，通过<logic:greaterEqual>标签判断 session 范围内指定的数值是否大于或等于要比较的常数值。

```
<%
    session.setAttribute("objectname",new Integer(123));
%>
<logic:greaterEqual value="100" name="objectname" scope="session">
    在 session 请求范围内 name 属性指定的数值大于或等于"100"！ <br>
</logic:greaterEqual>
```

16.4 <logic:greaterThan>标签——比较是否大于

<logic:greaterThan>标签用于比较指定变量是否大于指定常量。如果条件成立，Struts 会执行开始标签与结束标签之间的代码。

语法

```
<logic:greaterThan vlaue = “value” cookie = “cname” header = “head”
    name = “bname” parameter = “var” property = “bean” scope = “request”>
</logic:greaterThan>
```

<logic:greaterEqual>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例通过使用 parameter 属性指定的 HTTP 请求中参数值作为比较变量，判断给定常量是否大于变量，并输出相应信息。

```
<html:link href="?age=21">传递参数</html:link>
<br>
<logic:greaterThan value="23" parameter="age">
```

```
        传递的参数值大于或等于"23"！ <br>
    </logic:greaterThan>
    通过 parameter 属性指定的 HTTP 请求中的参数值作为比较数值。
```

16.5 <logic:lessEqual>标签——比较小于或等于

<logic:lessEqual>标签比较变量是否小于或等于指定的常量。如果条件成立，Struts 会执行开始标签与结束标签之间的代码。

语法

```
<logic:lessEqual vlaue = “value” cookie = “cname” header = “head”
    name = “bname” parameter = “var” property = “bean” scope = “request”>
</logic:lessEqual>
```

<logic:lessEqual>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例判断在 session 范围内的数值是否小于或等于 200，并给出提示信息。

```
<%
session.setAttribute("objectname", new Integer(123));
%>
<logic:lessEqual value="200" name="objectname" scope="session">
    在 session 会话范围内 name 属性指定的数值小于或等于“200”！ <br>
</logic:lessEqual>
```

16.6 <logic:lessThan>标签——比较是否小于

<logic:lessThan>标签用于判断变量是否小于指定的常量。如果条件成立，Struts 会执行开始标签与结束标签之间的代码。

语法

```
<logic:lessThan vlaue = “value” cookie = “cname” header = “head”
    name = “bname” parameter = “var” property = “bean” scope = “request”>
</logic:lessThan>
```

<logic:lessThan>标签属性如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域

示例

本示例使用<logic:lessThan>标签判断 session 中保存的变量是否小于指定常量，并给出提示信息。由于变量值为“100”不小于常量“100.0”，因此提示信息“100 is less than 100.0”不会输出。

```
<%
session.setAttribute("objectname", new Integer(100));
%>
<logic:lessThan value="100.0" name="objectname" scope="session">
        100 is less than 100.0
</logic:lessThan>
```

16.7 <logic:match>标签——匹配常量

<logic:match>标签用于判断变量中是否包含指定的常量字符串。如果变量中包含指定的常量字符串则返回结果为 true。

语法

```
<logic:match cookie = “cname” header = “head” parameter = “var”
             name = “bname” property = “pro” value = “value”
             scope = “request” location = “start”>
</logic:match>
```

<logic:match>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域
location	String	将该属性值设置为“start”，相当于 String 对象将 startsWith()；将该属性值设置为“end”，相当与 String 对象的 endsWith()方法实现的功能；忽略该属性时，相当于 String 对象的 indexOf()方法实现的功能

示例

本示例使用<logic:match>标签并将 location 属性设为“start”，判断指定变量的值是否以指定常量开头。

```
<%
    session.setAttribute("checkname1","Class1_yxq_man");
 %>
<logic:match value="Class1" name="checkname1" location="start">
    <bean:write name="checkname1"/> is class 1 student!
</logic:match>
```

本实例运行结果如图 16.1 所示。

示例

本示例省略<logic:match>标签中的 location 属性，判断指定变量中是否包含指定的常量。加载页面是，会将比较结果在页面中显示。

```
<%
session.setAttribute("checkname1", "Class1_yxq_man");
%>
<logic:match value="yxq" name="checkname1">
      yxq 存在 <bean:write name="checkname1"/> 中!
</logic:match>
```

本实例的运行结果如图 16.2 所示。

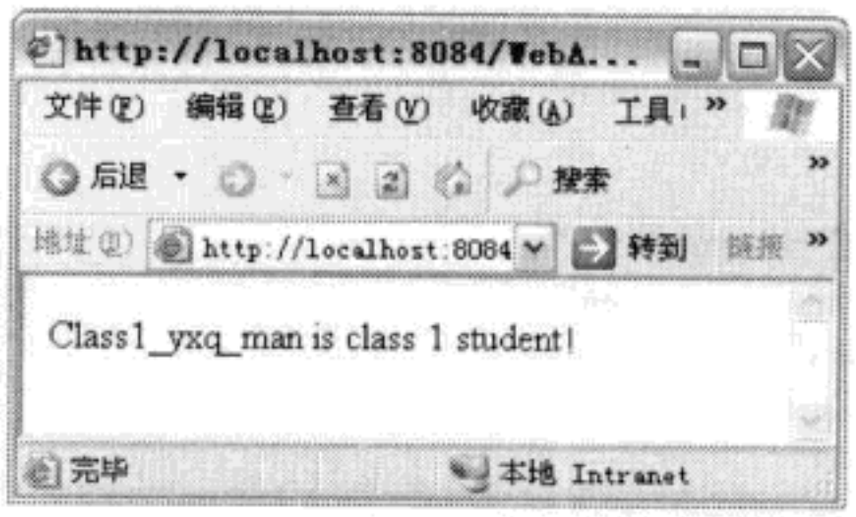


图 16.1 运行结果



图 16.2 运行结果

16.8 <logic:notMatch>标签——不匹配常量

<logic:noMatch>标签用于判断变量中是否不包含指定的常量字符串。如果变量中不包含指定的常量字符串，则返回结果为 true。

语法

```
<logic: noMatch cookie = “cname” header = “head” parameter = “var”
                name = “bname” property = “pro” value = “value”
                scope = “request” location = “start”>
</logic: noMatch>
```


PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

<logic:noMatch>标签属性如下表所示。

属　性	类　型	描　述
value	String	要做比较的常数值
cookie	String	要进行比较的 HTTP cookie 名
header	String	要进行比较的 HTTP 请求报头信息名
parameter	String	要进行比较的 HTTP 请求参数名
property	String	要进行比较的 bean 属性名
name	String	要进行比较的 bean 名称
scope	String	bean 的作用域
location	String	将该属性值设置为“start”，相当于 String 对象将 startsWith()；将该属性值设置为“end”，相当与 String 对象的 endsWith()方法实现的功能；忽略该属性时，相当于 String 对象的 indexOf()方法实现的功能

示例

本示例使用<logic:match>标签或<logic:notMatch>判断 request 范围内的变量“author”是否以指定的字符串开头，并给出相应的提示信息。

```
<%
request.setAttribute("author","LindaSun");
%>
<logic:notMatch name="author" scope="request"
                value="Linda" location="start">
    <bean:write name="author"/> doesn't start with the String 'Linda'
</logic:notMatch>
<logic:match name="author" scope="request" value="Linda" location="start">
    <bean:write name="author"/> starts with the String 'Linda'
</logic:match>
```

本实例运行结果如图 16.3 所示。



图 16.3　运行结果

16.9　<logic:empty>标签——对象为空

<logic:empty>标签用来判断在 request 或 sesssion 范围内指定的对象值是否为

空。如果指定的变量为 null 或者为空字符串“”，判断结果为 true。

语法

```
<logic:empty name = “object” property = “property” scope = “request”>
</logic:empty>
```

<logic:empty>标签属性如下表所示。

属　性	类　型	描　述
name	String	指定将要进行判断的对象
property	String	该属性匹配指定 JavaBean 中的某个属性值，指定属性值是否为空
scope	String	查找 name 属性指定对象的范围

示例

本示例使用<logic:empty>标签判断 session 范围内的对象是否为空，并给出相应的提示信息。

```
<%
   String name="";
   session.setAttribute("username",name);
%>
<logic:empty name="username" scope="session">
   用户名为空!
</logic:empty>
```

由于在 session 范围内的对象“username”为空，因此页面加载时提示信息“用户名为空”将会页面中显示。

16.10 <logic:notEmpty>标签——对象不为空

<logic:notEmpty>标签用来判断在 request 或 sesssion 范围内指定的对象值是否不为空。如果指定的变量不为 null，并且不为空字符串“”，则判断结果为 true。

语法

```
<logic:notEmpty name = “object”.property = “property” scope = “request”>
</logic:notEmpty>
```

<logic:notEmpty>标签属性如下表所示。

属　性	类　型	描　述
name	String	指定将要进行判断的对象
property	String	该属性匹配指定 JavaBean 中的某个属性值，指定属性值是否为空
scope	String	查找 name 属性指定对象的范围

示例

本示例判断 session 范围内的对象是否为空，并给出相应的信息。

PDF电子书基地 http://dayo1982.400gb.com

```
<%
String name = "mr";
String pwd = "";
session.setAttribute("username", name);
session.setAttribute("userpwd", pwd);
%>
<logic:empty name="userpwd" scope="session">
    用户密码为空!
</logic:empty>
<br>
<logic:notEmpty name="username" scope="session">
    用户名：<bean:write name="username"/>
</logic:notEmpty>
```

本示例运行结果如图 16.4 所示。

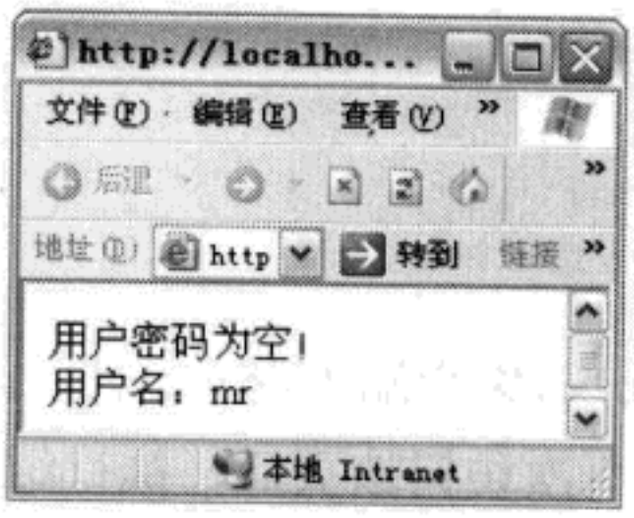


图 16.4 运行结果

16.11 <logic:present>标签——是否存在

<logic:present>标签用来判断指定的安全角色、用户、Cookie、HTTP 请求 Header 或 JavaBean 是否存在。

语法

```
<logic:present cookie = “cookieobject” header = “head” name = “object”
               parameter = “var” property = “proper” role = “role1”
               user = “username”>
</logic:present>
```

<logic:present>标签属性如下表所示。

属　性	类　型	描　述
cookie	String	指定了一个 Cookie 对象
header	String	该属性指定一个 HTTP 请求的标头信息
parameter	String	该属性指定了一个参数，标签将会判断该参数是否存在或不存在于请求中
name	String	指定一个已存在于 JSP 范围中的对象
property	String	指定 JavaBean 中的某个属性
role	String	指定安全角色
user	String	指定用户名

示例

本示例使用<logic:present>标签判断名称为“userName”的对象是否存在。

```
<%
request.setAttribute("userName", "Lina");
%>
<logic:present name="userName">
    <bean:write name="userName"/> is exits!
</logic:present>
```

示例

本示例判断指定的 cookie 对象是否存在。

```
<%
request.setAttribute("userName", "Lina");
%>
<logic:present name="userName">
    <bean:write name="userName"/> is exits!
</logic:present>
```

当第一次访问该页面后，若客户端不存在名为“userName”的 Cookie 对象，页面中无显示结果。当用户再次访问该页后，执行结果如图 16.5 所示。

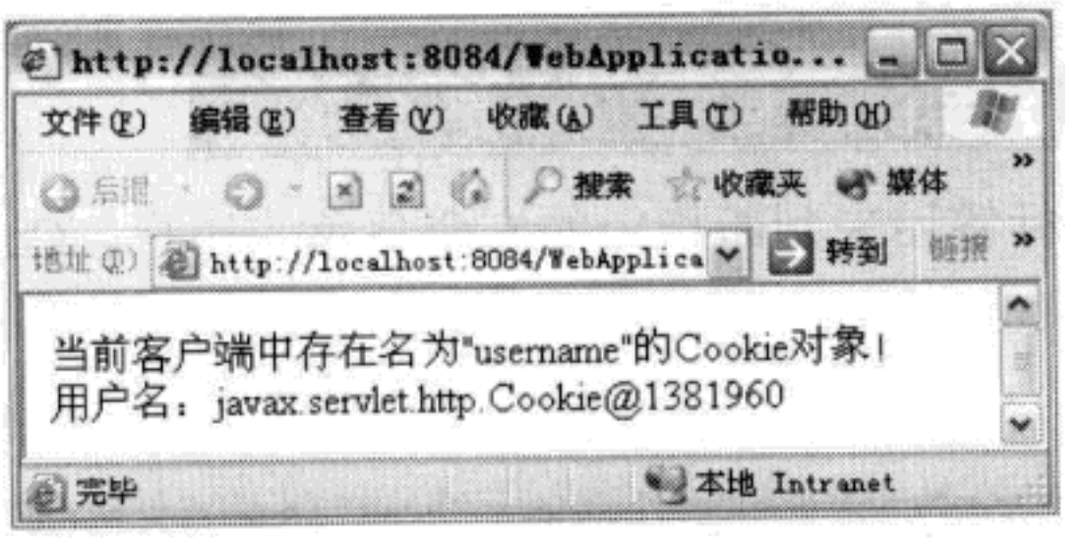


图 16.5 运行结果

16.12 <logic:notPresent>标签——不存在

<logic:notPresent>标签用于指定安全角色、用户、Cookie、HTTP 请求 Header 或 JavaBean 是否不存在，如果不存在则返回 true。

语法

```
<logic:notPresent cookie = “cookieobject” header = “head” name = “object”
                  parameter = “var” property = “proper” role = “role1”
                  user = “username”>
</logic:notPresent>
```

<logic:notPresent>标签属性如下表所示。

PDF电子书基地 http://dayo1982.400gb.com

属　性	类　型	描　述
cookie	String	指定了一个 Cookie 对象
header	String	该属性指定一个 HTTP 请求的标头信息
parameter	String	该属性指定了一个参数，标签将会判断该参数是否存在或不存在于请求中
name	String	指定一个已存在于 JSP 范围中的对象
property	String	指定 JavaBean 中的某个属性
role	String	指定安全角色
user	String	指定用户名

示例

本示例判断<logic:present>标签中 parameter 指定的参数是否存在，并给出相应的提示信息。

```
<html:link href="?myselect=A& myselect=B& myselect=C">传递参数
        </html:link><br>
<logic:notPresent parameter="myselect">
    当前请求中不存在名为“myselect”的参数！ <br>
</logic:notPresent>
<logic: parameter="myselect">
    当前请求中存在名为“myselect”的参数！ <br>
</logic:present>
```

16.13 <logic:messagesPresent>标签——存在指定消息

<logic:messagesPresent>标签用来判断 request 或 session 范围内是否存在指定的消息。

语法

```
<logic:messagesPresent name = “kname” property = “kpro”
                       message = “false”>
</logic:messagesPresent>
```

<logic:messagesPresent>标签属性如下表所示。

属　性	类　型	描　述
name	String	指定在 request 范围内检索 ActionMessages 对象的属性 key
property	String	指定从 ActionMessages 对象中检索某条消息的消息 key
message	String	如果为 true，则从 request 范围内检索属性 key 为 Globals.MESSAGE_KEY 的 ActionMessages 对象，此时 name 属性指定的值无效。 如果为 false 则根据 name 属性指定的属性 key，从 request 范围内检索 ActionMessages 对象，如果此时没有设置 name 属性，则默认的属性 key 为 Globals.ERROR_KEY

示例

本示例检索属性 key 为“s_message”的 ActionMessages 对象，如果判断结果为 true，则执行标签主体的内容。

```
<%
 ActionMessages messages = new ActionMessages();
 messages.add("lost",new ActionMessage("message1.title"));
 session.setAttribute("s_message",messages);
%>
<logic:messagesPresent name="s_message">
   存在 key 值为 s_message 的 ActionMessages 对象
</logic:messagesPresent>
```

16.14 <logic:messagesNotPresent>标签——不存在指定消息

<logic:messagesNotPresent>标签用来判断 request 或 session 范围内不存在指定的消息。如果不存在则返回 true。

语法

```
<logic:messagesNotPresent name = "kname" property = "kpro"
                          message = "false">
</logic:messagesNotPresent>
```

<logic:messagesPresent>标签属性如下表所示。

属　性	类　型	描　述
name	String	指定在 request 范围内检索 ActionMessages 对象的属性 key
property	String	指定从 ActionMessages 对象中检索某条消息的消息 key
message	String	如果为 true 则从 request 范围内检索属性 key 为 Globals.MESSAGE_KEY 的 ActionMessages 对象，此时 name 属性指定的值无效。 如果为 false 则根据 name 属性指定的属性 key，从 request 范围内检索 ActionMessages 对象，如果此时没有设置 name 属性，则默认的属性 key 为 Globals.ERROR_KEY

示例

本示例通过<logic:NotPresent>标签在 request 范围内检索属性 key 为 Globals.ERROR_KEY 的 ActionMessages 对象，然后再从 ActionMessages 对象中检索消息 key 为“noSuchError”的消息，如果不存在这样的信息，判断结果为 true，将执行标签主体的内容。

```
<logic:messagesNotPresent property="noSuchError">
    ActionMessages 对象中不存在 key 值是 noSuchError 的信息。
</logic:messagesNotPresent>
```

16.15 <logic:iterate>标签——遍历集合或数组

<logic:iterate>标签能够在一个循环中遍历数组、Collection、Enumeration、Iterator 或 Map 中的所有元素。该标签对给定的集合进行遍历，并以该集合中元素的数目为遍历次数。

PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

语法

```
<logic:iterate id = "var" collection = "coll" name = "bname"
               property = "property" indexed = "var" length = "len"
               offset = "start">
</logic:iterate>
```

<logic:iterate>标签属性如下表所示。

属　性	类　型	描　述
id	String	指定集合中遍历出的当前元素
collection	Object	指定一个集合
property	String	指定 bean 中属性名称
indexed	String	指定了一个存储了从指定集合中遍历出的当前元素序号的变量
length	int	指定遍历的次数
offset	int	指定了遍历集合的起始位置
name	String	该属性存储了需要被遍历的集合

示例

本示例实现将 Map 集合中的元素在当前集合元素的索引、Map 集合对应的 key 值、Map 集合中的 value 值在页面中显示。

```
<%
    HashMap months = new HashMap();
    months.put("Jan", "January");
    months.put("Feb", "February");
    months.put("Mar", "March");
    months.put("Apr", "April");
    months.put("May", "May");
    session.setAttribute("months",months);
%>
<logic:iterate id="element" indexId="ind" name="months">
    <bean:write name="element" property="key"/>:
    <bean:write name="element" property="value"/><br>
</logic:iterate>
```

本实例运行结果如图 16.6 所示。

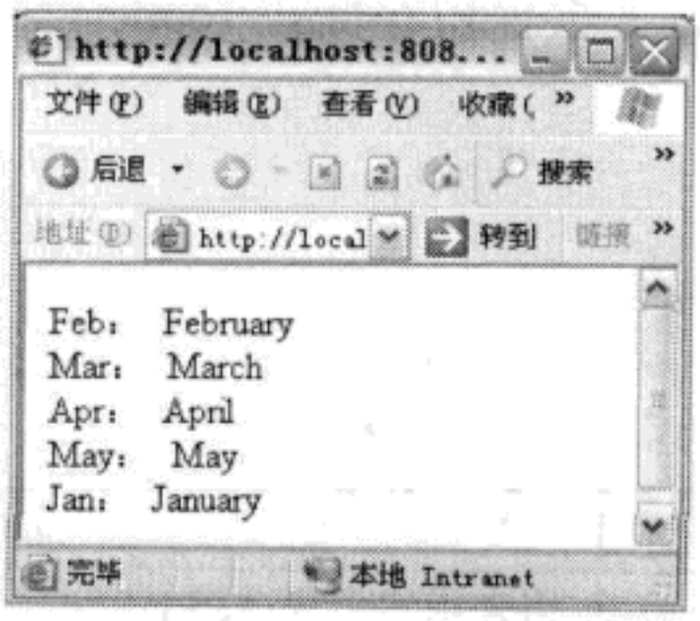


图 16.6　运行结果

示例

本示例定义了一个 Vector 类型的集合变量 vegetable，存放在 request 范围内。使用<logic:iterate>标签循环遍历集合中的所有元素。

```
<%
    Vector vegetable = new Vector();
    vegetable.addElement("aubergine");
    vegetable.addElement("tomato");
    vegetable.addElement("beanstalk");
    vegetable.addElement("cucumber");
    request.setAttribute("vegetable", vegetable);
%>
<logic:iterate id="element" indexId="ind" name="vegetable"
               offset="0" length="3">
    <b>
        <bean:write name="ind"/>：  <bean:write name="element"/><br>
    </b>
</logic:iterate>
```

本示例运行结果如图 16.7 所示。



图 16.7 运行结果

16.16 <logic:forward>标签——请求转发

<logic:forward>标签用于请求转发。<logic:forward>标签只能在当前应用中存在的资源间进行转发，该标签会保留存储在 request 对象中的信息并带到被转发至的页面。

语法

```
<logic:forward name = “aim”>
</logic:forward>
```

<logic:forward>标签属性如下表所示。

属　性	类　型	描　述
name	String	用来指定全局 ActionForward 的名称

PDF电子书基地 http://dayo1982.400gb.com

示例

若当前应用根目录下存在 index.jsp 和 welcome.jsp 两个文件。在 Struts 配置文件 struts-config.xml 中，设置全局转发代码如下：

```
<global-forwards>
  <forward name="welcome" path="/welcome.jsp"/>
</global-forwards>
```

在 index.jsp 页面中通过<logic:forward>标签实现用户访问 index.jsp 页面时，请求转发至 welcome.jsp 页面。

```
<body>
    <logic:forward name="welcome"/>
</body>
```

16.17 <logic:redirect>标签——请求重定向

<logic:redirect>标签用于请求重定向。使用<logic:redirect>标签可以重定向到外部的资源，而<logic:redirect>标签在客户端会生成新的 request 请求重定向到指定的 URL。

语法 1

```
<logic:redirect forward = “address”>
</logic:redirect>
```

语法 2

```
<logic:redirect href = “address”>
</logic:redirect>
```

语法 3

```
<logic:redirect page = “address”>
</logic:redirect>
```

<logic:redirect>标签属性如下表所示。

属　性	类　型	描　述
forward	String	映射了资源相对路径的 ActionForward
href	String	资源的完整 URL
redirect	String	资源的相对路径

示例

在 JSP 页面中编写如下代码，实现当访问该页面时，重定向至“welcome.jsp”页面。

```
<logic:redirect page="/welcome.jsp"/>
```

第 17 章　Hibernate 常用类与接口

17.1　Configuration 类——Hibernate 配置

Configuration 类位于 org.hibernate.cfg 包中，是 Hibernate 的入口，负责在 Hibernate 初始化时加载默认文件路径下的配置文件信息(hibernate.properties 或 hibernate.cfg.xml)到计算机内存，并通过它的对象加载指定的映射文件到内存，最后创建一个 SessionFactory 对象，把读入的配置信息复制到 SessionFactory 对象的缓存中。

说明

Configuration 类一般只有在获取 SessionFactory 时需要涉及，当 SessionFactory 创建后，由于配置信息已经由 Hibernate 绑定在返回的 SessionFactory 中，因此一般情况下无需再对其进行操作。

17.1.1　构造方法——Configuration 类的构造方法

Configuration 类的常用构造方法

构 造 方 法	使 用 说 明
Configuration()	创建 Configuration 类的对象，默认加载 classpah 中的配置文件 hibernate.properies

17.1.2　configure 方法——加载 Hibernate 配置文件

configure 方法用于告诉 Hibernate 加载 hibernate.cfg.xml 文件。默认情况下，Configuration 在实例化时加载 classpath 中的 hibernate.cfg.xml，当然也可以加载名称不是 hibernate.cfg.xml 的配置文件，这就需要应用 configure(URL url)方法。

方法声明 1

```
public Configuration configure()
                    throws HibernateException
```

返回值：负责管理 Hibernate 配置信息的 Configuration 对象。

抛出异常：HibernateException。

示例

创建一个 Configuration 对象，并通过该对象的 configure()方法加载 Hibernate 配置文件。

```
Configuration config=new Configuration().configure();
```


PDF电子书基地 http://dayo1982.400gb.com

方法声明 2

```
public Configuration configure(URL url)
                                throws HibernateException
```

入口参数：指定配置文件的 URL 路径。

返回值：通过指定文件构成的 Configuration 对象。

抛出异常：HibernateException。

示例

创建一个 Configuration 对象，并通过该对象的 configure 方法加载名称为 wghHibernate.cfg.xml 的配置文件。

```
Configuration config=new Configuration().configure("wghHibernate.cfg.xml");
```

17.1.3 addClass 方法——加载实体类

addClass 方法用于加载实体类。由于在通过从 Java 属性文件中读取配置信息构建 SessionFactory 时，缺少相应的配置映射文件的信息，所以在创建 Configuration 对象后，还需要调用 addClass 方法加载所需的实体类。

方法声明

```
public Configuration addClass(Class persistentClass)
                               throws MappingException
```

入口参数：需要加载的实体类。

返回值：负责管理 Hibernate 配置信息的 Configuration 对象。

抛出异常：MappingException。

示例

创建一个 Configuration 对象（此时 Hibernate 创建默认加载 classpath 中的配置文件 hibernate.properties），并且加载用户实体类 UserForm。

```
Configuration config=new Configuration();          //创建 Configuration 对象
Config.addClass(UserForm userForm);                //加载实体类 UserForm
```

17.1.4 buildSessionFactory 方法——构建 SessionFactory

buildSessionFactory 方法用于构建一个唯一的 SessionFactory，即 Session 实例的工厂，这个工厂将被应用程序的所有线程共享。

方法声明

```
public SessionFactory buildSessionFactory()
                                throws HibernateException
```

返回值：一个新的 SessionFactory 对象。

抛出异常：HibernateException。

示例

创建一个 Configuration 对象，然后调用 buildSessionFactory 方法创建一个 SessionFactory 对象。

```
Configuration config=new Configuration().configure();        //创建 Configuration 对象
SessionFactory sessionFactory=config.buildSessionFactory();
```

17.1.5 addResource 方法——指定实体类的映射文件

addResource 方法用于为 Configuation 对象指定实体类的 XML 映射文件。通常情况下不使用该方法，而是使用 addClass 方法让 Hibernate 根据实体类自动寻找对应的映射文件。

方法声明

```
public Configuration addResource(String resourceName)
                                        throws MappingException
```

入口参数：实体类的 XML 映射文件的路径。

返回值：Configuation 对象。

抛出异常：MappingException。

示例

创建一个 Configuration 对象，并为其指定实体类的 XML 映射文件。

```
Configuration cfg = new Configuration().addResource("UserForm.hbm.xml");
```

示例

创建一个 Configuration 对象，并为其指定多个实体类的 XML 映射文件。

```
Configuration cfg = new Configuration()
.addResource("UserForm.hbm.xml")
.addResource("TopicForm.hbm.xml");
```

17.1.6 setProperty 方法——为 Configuation 对象指定配置属性

setProperty 方法用于为 Configuation 对象指定配置属性。

方法声明

```
public Configuration setProperty(String propertyName,
                                 String value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：Configuation 对象。

示例

创建一个 Configuration 对象，并指定数据库使用的 SQL 语言。

```
Configuration cfg = new Configuration()
        .setProperty("hibernate.dialect", "org.hibernate.dialect.SQLServerDialect");
```


PDF电子书基地 http://dayo1982.400gb.com

示例

创建一个 Configuration 对象，并指定连接数据库的用户和密码。

```
Configuration cfg = new Configuration()
        .setProperty("connection.username ", "sa")
        .setProperty("connection.password", "111");
```

17.2 SessionFactory 接口——Session 工厂

SessionFactory 接口位于 org.hibernate 包中。SessionFactory 负责创建 Session 实例，每个 SessionFactory 类的实例对应一个数据库。SessionFactory 类的实例是重量级（重量级与轻量级是根据对象占用缓存的大小进行区分的）的，因为它占用很大的缓存，用来存放预定义的 SQL 语句和映射元数据等，所以每个数据库最好只创建一个 SessionFactory 实例，在初始化时完成；它也是线程安全的，可以被应用的多个线程共享。

在构建 SessionFactory 实例后，任何对 Configuration 实例进行的改变都不会影响到已经构建的 SessionFactory 实例，如果需要对 Configuration 实例进行的修改生效，就需要重新构建一个 SessionFactory 实例。

17.2.1 openSession 方法——开启一个 Session

openSession 方法用于通过 SessionFactory 创建 Session 实例，创建 Session 实例后，就可以通过创建的 Session 进行持久化操作了。

方法声明 1

创建数据库连接并打开一个 Session 实例。

```
public Session openSession()
                        throws HibernateException
```

返回值：Session 实例。

抛出异常：HibernateException。

示例

通过 SessionFactory 创建一个 Session 实例。

```
SessionFactory sessionFactory=config.buildSessionFactory();
Session session=sessionFactory.openSession();
```

方法声明 2

在指定的数据库连接中打开一个带有局部拦截器的 Session 实例。

```
public Session openSession(Connection connection,
                           Interceptor interceptor)
```

入口参数：connection 为一个数据库连接；interceptor 为一个 Session 范围的拦截器。

返回值：Session 实例。

示例

创建一个通过 JDBC 连接 SQL Server 数据库的连接，并在该数据库连接中打开一个带局部拦截器的 Session 实例。

```
Connection conn=null;
try {
    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
    String
url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_testHibernate";
    conn=DriverManager.getConnection(url,"sa","");
} catch (Exception e) {
    e.printStackTrace();
}
//MyIntereptor 为自定义的实现 Interceptor 接口的拦截器
Interceptor it=new MyInterceptor();
Session session=sessionFactory.openSession(conn,it);          //打开 Session
```

方法声明 3

创建数据库连接，并打开一个带有局部拦截器的 Session 实例。

```
public Session openSession(Interceptor interceptor)
    throws HibernateException
```

入口参数：一个 Session 范围的拦截器。

返回值：Session 实例。

抛出异常：HibernateException。

示例

通过 SessionFactory 创建一个带有局部栏截器的 Session 实例。

```
SessionFactory sessionFactory=config.buildSessionFactory();
//MyIntereptor 为自定义的实现 Interceptor 接口的拦截器
Interceptor it=new MyInterceptor();
Session session=sessionFactory.openSession(it);
```

方法声明 4

在给定的数据库连接中，打开一个 Session 实例。

```
public Session openSession(Connection connection)
```

入口参数：一个数据库连接。

返回值：Session 实例。

示例

在给定的数据库连接中，打开一个 Session 实例。

```
Connection conn=null;
try {
```


PDF电子书基地 http://dayo1982.400gb.com

```
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            String
url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_testHibernate";
            conn=DriverManager.getConnection(url,"sa","");
        } catch (Exception e) {
            e.printStackTrace();
        }
        Session session=sessionFactory.openSession(conn);          //打开 Session
```

17.2.2 isClosed 方法——SessionFactory 是否关闭

isClosed 方法用于判断 SessionFactory 对象是否关闭。

方法声明

```
public boolean isClosed()
```

返回值：表示判断结果的 true 或 false，为 true 表示已经关闭，为 false 表示未关闭。

示例

判断 SessionFactory 是否关闭，并输出判断结果。

```
if(sessionFactory.isClosed()){
        System.out.println("SessionFactory 已经关闭");
 }else{
        System.out.println("SessionFactory 未关闭");
 }
```

17.2.3 close 方法——关闭 SessionFactory 对象

close 方法用于关闭 SessionFactory 对象并释放所有的 SessionFactory 涉及的资源（缓存、连接池等），但在调用该方法前应该确保当前对象创建的 Session 全部被关闭。

方法声明

```
public void close()
            throws HibernateException)
```

抛出异常：HibernateException。

示例

首先创建 SessionFactory，然后当 SessionFactory 没有被关闭时将其关闭。

```
sessionFactory=config.buildSessionFactory();
if(sessionFactory.isClosed()){
        System.out.println("SessionFactory 已经关闭");
}else{
    sessionFactory.close();          //关闭 SessionFactory
}
```

17.3 Session 接口——持久化操作接口

Session 接口位于 org.hibernate 包中。Session 是 Hibernate 持久化操作的基础，它负责管理所有与持久化相关的操作，例如数据库的存取、事务的管理和对象的生命周期等。Session 与 SessionFactory 恰恰相反，它不是线程安全的，应避免多个线程共享同一个 Session；它是轻量级的，创建和销毁不会浪费太多的资源，可以为每个请求分配一个 Session，在每次请求过程中及时的创建和销毁 Session 实例，可以减少资源的浪费。

17.3.1 beginTransaction 方法——开启一个事务

beginTransaction 方法用于开启一个事务并返回一个 Transaction 对象。

方法声明

```
public Transaction beginTransaction()
                                    throws HibernateException
```

返回值：一个 Transaction 对象。

抛出异常：HibernateException。

示例

开启一个事务，并抛出异常信息。

```
Transaction tx=null;
try{
      tx=session.beginTransaction();   //开启事务
}catch(HibernateException e){
       e.printStackTrace();
}
```

17.3.2 isOpen 方法——判断 Session 是否开启

isOpen 方法用于判断 Session 对象是否被打开。

方法声明

```
public boolean isOpen()
```

返回值：表示判断结果的 true 或 false，为 true 表示已经打开，为 false 表示未打开。

示例

判断 Session 是否被打开，并输出判断结果。

```
if(session.isOpen()){
       System.out.println("Session 已经被打开");
 }else{
        System.out.println("Session 未被打开");
 }
```


PDF电子书基地 http://dayo1982.400gb.com

17.3.3 close 方法——关闭 Session

close 方法用于关闭一个 Session 对象。

方法声明

```
public Transaction beginTransaction()
                                    throws HibernateException
```

返回值：一个连接或是 null。

抛出异常：HibernateException。

示例

首先打开 Session，然后当 Session 被打开时将其关闭。

```
Session session=sessionFactory.openSession();
if(session.isOpen()){
    session.close();                    //关闭 Session
}
```

17.3.4 save 方法——保存对象

save 方法用于将一个临时对象转变为持久化对象，也就是将一个新的业务实体保存到数据库中。调用 save 方法后，对象的状态将由临时状态变为持久状态。

注意

在执行 save 方法前需要开启一个事务，并在执行结束后提交该事务。如果在执行 save 方法后，未提交事务，将不会访问数据库。

方法声明

```
public Serializable save(Object object)
                                throws HibernateException
```

入口参数：实体类的对象。

返回值：产生的标识符。

抛出异常：HibernateException。

示例

应用 save 方法将持久化对象保存到数据库，这里为添加用户信息。

```
Transaction tx=null;
try{
    tx=session.beginTransaction();      //开启事务
    UserForm uf=new UserForm();
    uf.setUsername(username);
    uf.setPwd(pwd);
    session.save(uf);                   //执行 save 方法，将持久化对象保存到数据库
    tx.commit();                        //提交事务
}catch(Exception e){
```

```
        if(tx!=null){
            tx.rollback();                    //回滚事务
        }
    }
```

17.3.5 update 方法——更新对象

update 方法用于将一个游离对象重新转变为持久化对象，也就是更新一个已经存在的业务实体到数据库中。

方法声明

```
public void update(Object object)
                throws HibernateException
```

入口参数：实体类的对象。

抛出异常：HibernateException。

示例

应用 update 方法将更新持久化对象，这里为修改用户信息。

```
Transaction tx=null;
try{
    tx=session.beginTransaction();              //开启事务
    UserForm uf=new UserForm();
    uf.setPwd(pwd);
    session.update(uf);
    tx.commit();                                //提交事务
}catch(Exception e){
    if(tx!=null){
        tx.rollback();                          //回滚事务
    }
}
```

17.3.6 saveOrUpdate 方法——保存或更新对象

saveOrUpdate 方法用于根据传入参数的状态执行不同的操作，当为临时状态时，调用 save()方法；当为持久化状态时，则直接返回；当为游离状态时，调用 update()方法。

方法声明

```
public void saveOrUpdate(Object object)
                throws HibernateException
```

入口参数：实体类的对象，用于指定将哪个持久化对象保存或更新到数据库中。

抛出异常：HibernateException。

示例

应用 saveOrUpdate 方法保存或更新持久化对象到数据库，这里为保存或更

PDF电子书基地 http://dayo1982.400gb.com

新用户信息。

```
Session session=sessionFactory.openSession();
String username="sk";
String pwd="111";
Transaction tx=null;
try{
    tx=session.beginTransaction();                    //开启事务
    UserForm uf=new UserForm();
    uf.setUsername(username);
    uf.setPwd(pwd);
    session.saveOrUpdate(uf);                         //调用 saveOrUpdate 方法保存数据
    tx.commit();                                      //提交事务
}catch(Exception e){
    if(tx!=null){
        tx.rollback();                                //回滚事务
    }
}
```

17.3.7 delete 方法——删除对象

delete(Object object)方法的作用是删除与传入的持久化类对象对应的数据库当中的记录，传入的持久化类对象可以是持久化状态的，也可以是游离状态的，但不可以是临时状态的，如果是持久化状态的，则在销毁 Session 清理缓存时执行 delete 语句；如果是游离状态的，首先是将游离状态的对象转变为持久化状态，然后在销毁 Session 清理缓存时执行 delete 语句。

方法声明

```
public void delete(Object object)
        throws HibernateException
```

入口参数：实体类的对象，用于指定删除哪个持久化对象中的数据。

抛出异常：HibernateException。

示例

应用 delete 方法删除对象，这里为删除指定的一条用户信息。

```
Session session=sessionFactory.openSession();
Transaction tx=null;
try{
    tx=session.beginTransaction();                          //开启事务
    // 通过 HQL 查询创建一个处于持久化状态的对象
    UserForm uf=(UserForm)session.createQuery("from UserForm where username =' s k
'").setMaxResults(1).uniqueResult();
    session.delete(uf);                                     //删除指定的用户信息
    tx.commit();                                            //提交事务
}catch(Exception e){
    if(tx!=null){
```

```
            tx.rollback();                                    //回滚事务
        }
    }
```

示例

批量删除用户名为 sk 的数据。

```
Session session=sessionFactory.openSession();
Transaction tx=null;
try{
    tx=session.beginTransaction();                            //开始事务
    List list=session.createQuery("from UserForm where username='sk'").list();
    Iterator it=list.iterator();
    while(it.hasNext()){
        UserForm uf=(UserForm)it.next();                      //获取一个对象
        session.delete(uf);                                   //删除指定对象
    }
    tx.commit();                                              //提交事务
}catch(Exception e){
    if(tx!=null){
        tx.rollback();                                        //回滚事务
    }
}
```

17.3.8 get 方法——查询对象

Session 的 get()方法用于通过指定的 OID 从数据库中加载一个实体对象。在应用该方法时，当数据库中不存在与指定的 OID 对应的持久化对象时，将返回 null。该方法总是立即加载对象。

方法声明 1

```
public Object get(Class clazz, Serializable id)
                throws HibernateException
```

入口参数：clazz 为实体类；id 用于指定 OID。

返回值：实体类的对象或是 null。

抛出异常：HibernateException。

示例

应用 get 方法查询 OID 为 1 用户信息。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.get(UserForm.class, 1);
```

方法声明 2

```
public Object get(String entityName, Serializable id)
                throws HibernateException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：entityName 为表示实体类的字符串（这里需要指定对应的包）；id 用于指定 OID。

返回值：实体类的对象或是 null。

抛出异常：HibernateException。

示例

应用 get 方法查询 OID 为 1 用户信息。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.get("com.model.UserForm",1);
```

方法声明 3

```
public Object get(Class clazz, Serializable id, LockMode lockMode)
                throws HibernateException
```

入口参数：clazz 为实体类；id 用于指定 OID；lockMode 用于指定锁定模式。

LockMode 类提供的几种锁定模式

锁 定 模 式	功 能 简 介
LockMode.NONE	表示无锁机制，是 Hibernate 的默认值。采用该种方式首先在 Hibernate 缓存中检索，存在则直接返回对象的引用；否则到数据库中加载对象。是 Hibernate 的默认值
LockMode.READ	无论 Hibernate 缓存中是否存在，总是直接到数据库中加载。如果在映射文件中设置了版本元素，则比较缓存中对象的版本与数据库中记录的版本是否一致
LockMode.UPGRADE	在 LockMode.READ 模式的基础上，如果数据库系统支持悲观锁（如 MySQL 和 Oracle），则执行 select……for update 语句，否则执行普通的 select 语句
LockMode.UPGRADE_NOWAIT	在上一模式的基础上，如果是 Oracle 数据库，则执行 select……for update nowait 语句，“nowait”的作用是如果执行该 select 语句的事务不能立刻获得悲观锁，会立刻抛出锁定异常，而不会去等待其他事务释放锁定

返回值：实体类的对象或代理对象。

抛出异常：HibernateException。

示例

应用 get 方法查询 OID 为 1 的用户信息，并使用 LockMode.READ 模式锁定获取的对象。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.get("com.model.UserForm",1, LockMode.READ);
```

17.3.9 load 方法——加载对象

Session 的 load()方法用于通过指定的 OID 从数据库中加载一个实体对象。在应用该方法时，当数据库中不存在与指定的 OID 对应的持久化对象时，将抛出异常，所以在采用 load()方法加载对象时一定要保证该对象的存在，否则最好用 get()方法。该方法支持延迟检索策略，当采用延迟检索策略时，在执行 load()方法时只是为该对象创建了一个代理对象，仅初始化它的 OID，只有在该对象被访问时，才执行 select 语句，初始化该代理对象的其他信息。

方法声明 1

```
public Object load(Class theClass, Serializable id)
                throws HibernateException
```

入口参数：theClass 为实体类；id 用于指定 OID。

返回值：实体类的对象或代理对象。

抛出异常：HibernateException。

示例

应用 load 方法查询 OID 为 1 用户信息。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.load(UserForm.class, 1);
```

方法声明 2

```
public Object load(String entityName, Serializable id)
                throws HibernateException
```

入口参数：entityName 为表示实体类的字符串（这里需要指定对应的包）；id 用于指定 OID。

返回值：实体类的对象或代理对象。

抛出异常：HibernateException。

示例

应用 load 方法查询 OID 为 1 用户信息。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.load("com.model.UserForm",1);
```

方法声明 3

```
public Object load(Class theClass, Serializable id, LockMode lockMode)
                throws HibernateException
```

入口参数：theClass 为实体类；id 用于指定 OID；lockMode 用于指定锁定模式。

PDF电子书基地 http://dayo1982.400gb.com

LockMode 类提供的几种锁定模式

锁 定 模 式	功 能 简 介
LockMode.NONE	表示无锁机制，是 Hibernate 的默认值。采用该种方式首先在 Hibernate 缓存中检索，存在则直接返回对象的引用；否则到数据库中加载对象。是 Hibernate 的默认值
LockMode.READ	无论 Hibernate 缓存中是否存在，总是直接到数据库中加载。如果在映射文件中设置了版本元素，则比较缓存中对象的版本与数据库中记录的版本是否一致
LockMode.UPGRADE	在 LockMode.READ 模式的基础上，如果数据库系统支持悲观锁（如 MySQL 和 Oracle），则执行 select……for update 语句，否则执行普通的 select 语句
LockMode.UPGRADE_NOWAIT	在上一模式的基础上，如果是 Oracle 数据库，则执行 select……for update nowait 语句，“nowait”的作用是如果执行该 select 语句的事务不能立刻获得悲观锁，会立刻抛出锁定异常，而不会去等待其他事务释放锁定

返回值：实体类的对象或代理对象。

抛出异常：HibernateException。

示例

应用 load 方法查询 OID 为 1 的用户信息，并使用 LockMode.READ 模式锁定获取的对象。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.load("com.model.UserForm",1, LockMode.READ);
```

17.3.10 evict 方法——从 Session 缓存中清除对象

evict()方法用于将指定的持久化对象从 Session 缓存中清除，是 Hibernate 提供管理缓存的方法。

方法声明

```
public void evict(Object object)
                throws HibernateException
```

入口参数：实体类的对象。

抛出异常：HibernateException。

示例

应用 evict 方法将缓存中的用户对象清除。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.get("com.model.UserForm", 1);
System.out.println("执行 evict 方法前，uf 是否在缓存中："+session.contains(uf));
 session.evict(uf);                          //执行 evict 方法
 System.out.println("执行 evict 方法后，uf 是否在缓存中："+session.contains(uf));
```

执行上面的代码，在控制台中将显示以下内容：

```
执行 evict 方法前，uf 是否在缓存中：true
执行 evict 方法后，uf 是否在缓存中：false
```

17.3.11 clear 方法——清除 Session 缓存中的所有所有对象

clear 方法用于清空缓存，即清除 Session 缓存中的所有持久化对象。当缓存中存放一定数量的持久化对象时，就会抛出缓存不足的异常。

方法声明

```
public void clear()
```

示例

应用 clear 方法清除 Session 缓存中的所有持久化对象。

```
Session session=sessionFactory.openSession();
UserForm uf=new UserForm();
uf=(UserForm)session.get("com.model.UserForm", 1);
System.out.println("执行 clear 方法前，uf 是否在缓存中："+session.contains(uf));
session.clear();
System.out.println("执行 clear 方法后，uf 是否在缓存中："+session.contains(uf));
```

执行上面的代码，在控制台中将显示以下内容：

```
执行 clear 方法前，uf 是否在缓存中：true
执行 clear 方法后，uf 是否在缓存中：false
```

17.3.12 contains 方法——判断对象是否在缓存中存在

contains 方法用来判断指定的实体类的对象是否在缓存中，它的返回值为 boolean 型，如果返回值为 true，则说明在缓存中；如果为 false 则不在缓存中。

方法声明

```
public boolean contains(Object object)
```

入口参数：实体类的对象。

返回值：表示判断结果的 true 或 false，为 true 表示在缓存 ，为 false 表示不在缓存中。

示例

应用 contains 方法判断指定的实体类对象 uf 是否在缓存中。

```
if(session.contains(uf)){
    System.out.println("uf 在缓存中");
}else{
    System.out.println("uf 不在缓存中");
}
```

17.3.13 createQuery 方法——创建 Query 对象

createQuery 方法用来生成一个基于 HQL 查询字符串的 Query 对象。

PDF电子书基地 http://dayo1982.400gb.com



方法声明

```
public Query createQuery(String queryString)
                        throws HibernateException
```

入口参数：HQL 查询字符串。

返回值：Query 对象。

抛出异常：HibernateException。

示例

应用 HQL 查询全部用户信息。

```
Session session=sessionFactory.openSession();
 String hql="from UserForm";                                    //指定 HQL 语句
 List list=null;
 try{
       Query query=session.createQuery(hql);                    //生成 Query 对象
       list=query.list();
 }catch(Exception e){
       System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.3.14 createSQLQuery 方法——创建 SQLQuery 对象

createSQLQuery 方法用来生成一个基于 SQL 查询字符串的 Query 对象。通过该方法可以直接使用数据库 SQL 语言进行查询，并且对查询结果进行了简单的封装，不必再对 ResultSet 进行烦琐的操作，因为查询结果依然是由持久化对象组成的集合，仍然可以用面向对象的思想进行编程。

方法声明

```
public SQLQuery createSQLQuery(String queryString)
                              throws HibernateException
```

入口参数：SQL 查询字符串。

返回值：Query 对象。

抛出异常：HibernateException。

示例

应用 SQL 查询全部用户信息，并保存到 List 集合中。

```
Session session=sessionFactory.openSession();
String sql="select * from tb_user";                             //指定 SQL 语句
List list=null;
List rtnList=new ArrayList();
try{
     Query query=session.createSQLQuery(sql);                   //执行 SQL 语句
     Iterator it =query.list().iterator();
```

```
while(it.hasNext()){                          //通过循环将查询到的用户信息保存到 List 集合中
    UserForm uf=new UserForm();
    Object[] o=(Object[])it.next();
    uf.setId((Integer)o[0]);
    uf.setUsername((String)o[1]);
    uf.setPwd((String)o[2]);
    rtnList.add(uf);
}
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.3.15 createCriteria 方法——创建 Criteria 对象

createCriteria 方法用来生成一个 Criteria 对象，可以返回符合检索条件的该类对象的实例集合。

方法声明

```
public Criteria createCriteria(Class persistentClass)
```

入口参数：实体类。

返回值：Criteria 对象。

示例

应用 QBC 查询全部用户信息。

```
Session session=sessionFactory.openSession();          //打开 Session
Criteria criteria=session.createCriteria(UserForm.class);    //生成 Criteria 对象
List list=criteria.list();
```

17.3.16 lock 方法——指定的锁定模式进行锁定

lock 方法用来将指定的实体类对象以指定的锁定模式进行锁定。

方法声明

```
public void lock(Object object,LockMode lockMode)
        throws HibernateException
```

入口参数：object 为一个实体类的对象；lockMode 为锁定模式，可用的值如下表所示。

LockMode 类提供的几种锁定模式

锁 定 模 式	功 能 简 介
LockMode.NONE	表示无锁机制，是 Hibernate 的默认值。采用该种方式首先在 Hibernate 缓存中检索，存在则直接返回对象的引用；否则到数据库中加载对象。是 Hibernate 的默认值
LockMode.READ	无论 Hibernate 缓存中是否存在，总是直接到数据库中加载。如果在映射文件中设置了版本元素，则比较缓存中对象的版本与数据库中记录的版本是否一致

PDF电子书基地 http://dayo1982.400gb.com

续表

锁 定 模 式	功 能 简 介
LockMode.UPGRADE	在 LockMode.READ 模式的基础上，如果数据库系统支持悲观锁（如 MySQL 和 Oracle），则执行 select……for update 语句，否则执行普通的 select 语句
LockMode.UPGRADE_NOWAIT	在上一模式的基础上，如果是 Oracle 数据库，则执行 select……for update nowait 语句，“nowait”的作用是如果执行该 select 语句的事务不能立刻获得悲观锁，会立刻抛出锁定异常，而不会去等待其他事务释放锁定

抛出异常：HibernateException。

示例

应用 get 方法获取实体类的对象，并使用 LockMode.READ 模式锁定该对象。

```
UserForm uf=new UserForm();
uf=(UserForm)session.get(UserForm.class, 1);
session.lock(uf,LockMode.READ);
```

17.4 Transaction 接口——事物接口

Transaction 接口位于 org.hibernate 包中。Transaction 类负责 Hibernate 的数据库事务，其实 Hibernate 本身并不具有管理事务的能力，只是对底层事务接口进行了封装，这样做的好处是有利于应用在不同的环境或容器中移植，也可以直接访问底层的事务接口，但是笔者不赞成，那样不利于应用的移植。

17.4.1 begin 方法——开启事务

begin 方法用于开启事务。

方法声明

```
public void begin()
        throws HibernateException
```

抛出异常：HibernateException。

示例

应用 begin 方法开启事务。

```
Transaction tx=session.getTransaction();
tx.begin();
```

17.4.2 commit 方法——提交事务

commit 方法用于提交事务。

方法声明

```
public void commit()
                throws HibernateException
```

抛出异常：HibernateException。

示例

应用 commit 方法提交事务。

```
Session session=sessionFactory.openSession();
Transaction tx=session.beginTransaction();              //开启事务
 UserForm uf=new UserForm();
 uf.setUsername(username);
 uf.setPwd(pwd);
 session.saveOrUpdate(uf);
 tx.commit();                                           //提交事务
```

17.4.3 rollback 方法——回滚事务

rollback 方法用于回滚事务。在进行数据库操作时，当程度出现错误时，及时进行事务回滚，可以保证数据的有效性。

方法声明

```
public void rollback()
                throws HibernateException
```

抛出异常：HibernateException。

示例

当添加数据抛出异常时，应用 rollback 方法回滚事务。

```
Session session=sessionFactory.openSession();
Transaction tx=session.beginTransaction();              //开启事务
 UserForm uf=new UserForm();
 uf.setUsername(username);
 uf.setPwd(pwd);
 session.saveOrUpdate(uf);
 tx.commit();                                           //提交事务
```

17.4.4 wasCommitted 方法——判断事务是否被提交

wasCommitted 方法用于判断事务是否被提交。

方法声明

```
public boolean wasCommitted()
                        throws HibernateException
```

返回值：表示判断结果的 true 或是 false，为 true 时，表示已经提交，否则表示未提交。

抛出异常：HibernateException。

PDF电子书基地 http://dayo1982.400gb.com



示例

当添加数据抛出异常时，应用 rollback 方法回滚事务。

```
session.saveOrUpdate(uf);
if(tx.wasCommitted()){
      System.out.println("事务已经被提交");
}else{
     System.out.println("事务没有被提交");
}
tx.commit();                                          //提交事务
if(tx.wasCommitted()){
      System.out.println("事务已经被提交");
}else{
     System.out.println("事务没有被提交");
}
```

执行上面的代码，在控制台将显示以下内容：

```
事务没有被提交
Hibernate: insert into tb_user (username, pwd, id) values (?, ?, ?)
事务已经被提交
```

17.4.5 wasRolledBack 方法——判断事务是否回滚

wasRolledBack 方法用于判断事务是否回滚。

方法声明

```
public boolean wasRolledBack()
                        throws HibernateException
```

返回值：表示判断结果的 true 或是 false，为 true 时，表示已经回滚，否则表示未回滚。

抛出异常：HibernateException。

示例

当添加数据抛出异常时，应用 rollback 方法回滚事务。

```
try{
     tx=session.beginTransaction();                   //开启事务
     UserForm uf=new UserForm();
     uf.setUsername(username);
     uf.setPwd(pwd);
     session.saveOrUpdate(uf);
     tx.commit();                                     //提交事务
}catch(Exception e){
     if(tx!=null){
          if(tx.wasRolledBack()){
               System.out.println("事务已经回滚");
          }else{
               System.out.println("事务没有回滚");
```

```
        }
        tx.rollback();                                          //回滚事务
         if(tx.wasRolledBack()){
              System.out.println("事务已经回滚");
        }else{
             System.out.println("事务没有回滚");
        }
    }
```

执行上面的代码，当产生异常信息时，将在控制台显示以下内容：

```
事务没有回滚
事务已经回滚
```

17.5 Query 接口——HQL 查询接口

Query 接口位于 org.hibernate 包中。Query 接口负责 Hibernate 的查询操作。Query 实例封装了一个 HQL（Hibernate Query Language）查询语句，HQL 与 SQL 有些类似，只是 HQL 是面向对象的，它操作的是持久化类的类名和该类的属性名，而 SQL 操作的是表名和字段名。

17.5.1 list 方法——返回查询结果的 List 集合

list 方法用来返回一个保存查询结果的 List 集合，该集合可能是由 Object 对象组成的，也可能是由 Object 类型数组组成。

方法声明

```
public List list()
            throws HibernateException
```

返回值：List 集合。

抛出异常：HibernateException。

示例

在进行 HQL 查询时，调用 list()方法返回保存查询结果的 List 集合。

```
Session session=sessionFactory.openSession();
String hql="from UserForm";
List list=null;
try{
    Query query=session.createQuery(hql);
    list=query.list();                          //调用 list()方法返回一个 List 集合
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.5.2 iterate 方法——返回查询结果的迭代器

iterate 方法用来返回一个按照从前向后的顺序依次读取数据库记录的

PDF电子书基地 http://dayo1982.400gb.com

Iterator 对象。该方法的优点是不用一次读取所有数据并转换为 Java 对象，缺点是只能按顺序来访问查询结果集，而且不知道结果集中记录的数目。

方法声明

```
public Iterator iterate()
                    throws HibernateException
```

返回值：Iterator 对象。

抛出异常：HibernateException。

示例

在进行 HQL 查询时，调用 iterator 方法返回读取数据库记录的 Iterator 对象，并将读取的结果保存到 List 集合中。

```
Session session=sessionFactory.openSession();
String hql="from UserForm";
Iterator it=null;
List rtnList=new ArrayList();
try{
    Query query=session.createQuery(hql);
    it=query.iterate();                          //返回读取数据库记录的 Iterator 对象
    while(it.hasNext()){
        UserForm uf=new UserForm();
        Object[] o=(Object[])it.next();
        uf.setId((Integer)o[0]);
        uf.setUsername((String)o[1]);
        uf.setPwd((String)o[2]);
        rtnList.add(uf);                         //添加到 List 集合中
    }
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.5.3 uniqueResult 方法——单值检索

uniqueResult 方法用于返回唯一的结果对象。该方法一般在确信只有一个满足条件的记录时使用，如果结果集的记录条数超过将会抛出异常。

方法声明

```
public Object uniqueResult()
                    throws HibernateException
```

返回值：唯一的结果对象或是 null。

抛出异常：HibernateException。

示例

获取用户信息表中的第一条记录。

```
Session session=sessionFactory.openSession(conn);
 String hql="from UserForm";
 List list=new ArrayList();
 try{
      Query query=session.createQuery(hql);
      UserForm uf=(UserForm)query.setMaxResults(1).uniqueResult();
      list.add(uf);
  }catch(Exception e){
      System.out.println("查询用户信息时的错误信息："+e.getMessage());
  }
```

17.5.4 setMaxResults 方法——设置最大返回记录数

setMaxResults 方法用于指定返回的 Query 对象中最大的记录条数，该方法通常用在数据分页时。

方法声明

```
public Query setMaxResults(int maxResults)
```

入口参数：用于指定返回的 Query 对象中最大的记录条数。

返回值：Query 对象。

示例

获取用户信息表中的前 5 条记录。

```
Session session=sessionFactory.openSession(conn);
String hql="from UserForm";
List list=null;
try{
      Query query=session.createQuery(hql);
      list=query.setMaxResults(5).list();
}catch(Exception e){
      System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.5.5 setFirsResult 方法——设置结果集的起始点

setFirsResult 方法用于指定返回的结果集的起始点，该方法通常用在数据分页时。

方法声明

```
public Query setFirstResult(int firstResult)
```

入口参数：用于指定返回结果集的起始点。

返回值：Query 对象。

示例

获取用户信息表中第 2 条记录以后的全部数据，但不包括第 2 条。

PDF电子书基地 http://dayo1982.400gb.com

```
Session session=sessionFactory.openSession();
String hql="from UserForm";
List list=null;
try{
      Query query=session.createQuery(hql);
      list=query.setFirstResult(2).list();
}catch(Exception e){
      System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.5.6 setEntity 方法——用于绑定实体类参数

setEntity 方法用于绑定实体类参数。

方法声明

```
public Query setEntity(String name,
                       Object val)
```

入口参数：name 用于指定实体类的类名，val 用于指定实体类对象。

返回值：Query 对象。

示例

查询部门编号为 3 的员工信息。

```
Dept d = new Dept();
d.setId(3);
Query q = session.createQuery("from Staff where dept_id=:DEPT");
q.setEntity("DEPT", d);
```

在上面的代码中，是不可以通过部门名称检索的，之所以可以通过部门 id 检索是因为在本例中为两个类建立了一对多的关联关系，具体是通过员工的外键（dept_id）参照部门主键（id）实现的。

17.5.7 setParameter 方法——绑定任意类型的参数

setParameter 方法用于绑定任意类型的参数。

方法声明

```
public Query setParameter(String name,Object val)
                          throws HibernateException
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

抛出异常：HibernateException。

示例

通过参数动态绑定机制实现查询用户名为 wgh 的用户信息。

```
String hql="from UserForm where username=:userName";
Query query=session.createQuery(hql);
query.setParameter("userName","wgh");
list=query.list();
```

17.5.8 setDate 方法——绑定映射类型为 Date 的参数

setDate 方法用于绑定映射类型为 Date 的参数。

方法声明 1

```
public Query setDate(int position, Date date)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询生日为 1980-7-17 的用户信息。

```
String hql="from UserForm where birthday=?";
Query query=session.createQuery(hql);
query.setDate(0,Date.valueOf("1980-7-17"));
```

方法声明 2

```
public Query setDate(String name, Date date)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询生日为 1980-7-17 的用户信息。

```
String hql="from UserForm,where birthday=:birthday";
Query query=session.createQuery(hql);
query.setDate("birthday", Date.valueOf("1980-7-17"));
list=query.list();
```

17.5.9 setString 方法——绑定映射类型为 String 的参数

setString 方法用于绑定映射类型为 String 的参数。

方法声明 1

```
public Query setString(int position,
                       String val)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询性别为“女”的用户信息。

PDF电子书基地 http://dayo1982.400gb.com

```
String hql="from UserForm where sex=?";
Query query=session.createQuery(hql);
query.setString(0, "女");
```

方法声明 2

```
public Query setString(String name, String val)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询性别为“女”的用户信息。

```
String hql="from UserForm where sex=:sex";
Query query=session.createQuery(hql);
query.setString("sex","女");
```

17.5.10 setInteger 方法——绑定映射类型为 int 的参数

setInteger 方法用于绑定映射类型为 int 的参数。

方法声明 1

```
public Query setInteger(int position,
                        int val)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询 id 为“6”的用户信息。

```
String hql="from UserForm where id=?";
Query query=session.createQuery(hql);
query.setInteger(0,6);
```

方法声明 2

```
public Query setInteger(String name,
                        int val)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询 id 为 6 的用户信息。

```
String hql="from UserForm where id=:id";
Query query=session.createQuery(hql);
query.setInteger("id",6);
```

17.5.11 setBoolean 方法——绑定映射类型为 boolean 的参数

setBoolean 方法用于绑定映射类型为 boolean 的参数。

方法声明 1

```
public Query setBoolean(int position,
                        boolean val)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询删除标记为 true 的用户信息。

```
String hql="from UserForm where isDel=?";
Query query=session.createQuery(hql);
query.setBoolean(0,true);
```

方法声明 2

```
public Query setBoolean(String name,
                        boolean val)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询删除标记为 true 的用户信息。

```
String hql="from UserForm where isDel=:isDel";
Query query=session.createQuery(hql);
query.setBoolean("isDel",true);
```

17.5.12 setDouble 方法——绑定映射类型为 double 的参数

setDouble 方法用于绑定映射类型为 double 的参数。

方法声明 1

```
public Query setDouble(int position,
                       double val)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询单价为 5967.2 的商品信息。

```
String hql="from GoodsForm where price=?";
Query query=session.createQuery(hql);
query.setDouble(0,5967.2);
```


PDF电子书基地 http://dayo1982.400gb.com



方法声明 2

```
public Query setDouble(String name,
                       double val)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询单价为 5967.2 的商品信息。

```
String hql="from GoodsForm where price=:price";
Query query=session.createQuery(hql);
query.setDouble("price",5967.2);
```

17.5.13 setFloat 方法——绑定映射类型为 float 的参数

setFloat 方法用于绑定映射类型为 float 的参数。

方法声明 1

```
public Query setFloat(int position,
                      float val)
```

入口参数：position 用于指定是第几个参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用顺序占位符方式实现查询单价为 29.8 的商品信息。

```
String hql="from GoodsForm where price=?";
Query query=session.createQuery(hql);
query.setFloat(0,29.8);
```

方法声明 2

```
public Query setFloat(String name,
                      float val)
```

入口参数：name 用于指定在 HQL 语句中定义的参数；val 用于指定对应的参数值。

返回值：Query 对象。

示例

使用引用占位符方式实现查询单价为 29.8 的商品信息。

```
String hql="from GoodsForm where price=:price";
Query query=session.createQuery(hql);
query.setFloat("price",29.8);
```

17.6 Criteria 接口——QBC 查询接口

Criteria 接口位于 org.hibernate 包中。Criteria 接口负责 Hibernate 的查询操作。Criteria 实例完全封装了字符串形式的查询语句，它比 Query 实例要面向对象，它更适合于执行动态查询。

17.6.1 add 方法——设置查询条件

add 方法用于设置查询条件。

方法声明

```
public Criteria add(Criterion criterion)
```

入口参数：为 Criterion 对象的实例，可以通过 Restrictions 类或者 Expression 类所提供的一系列静态方法来创建。

返回值：Criteria 对象。

示例

应用 QBC 查询，查询用户名为 sk 的全部用户信息。

```
Session session=sessionFactory.openSession();
List list=null;
try{
    Criteria criteria=session.createCriteria(UserForm.class);
    criteria.add(Restrictions.eq("username","sk"));
    list=criteria.list();
}catch(Exception e){
        System.out.println("查询用户信息时的错误信息：  "+e.getMessage());
}
```

17.6.2 addOrder 方法——设置结果集的排序规则

addOrder 方法用于设置结果集的排序规则。

方法声明

```
public Criteria addOrder(Order order)
```

入口参数：为 Order 对象的实例，可以通过 Order 类的 desc 方法（降序方法）和 asc 方法（升序方法）指定等排序方式。

返回值：Criteria 对象。

示例

应用 QBC 查询，对主题信息按发表时间降序排列。

```
Session session=sessionFactory.openSession();
Criteria criteria = session.createCriteria(TbTopic.class);
criteria.addOrder(Order.desc("createTime"));
List list=criteria.list();
```


PDF电子书基地 http://dayo1982.400gb.com

17.6.3 createCriteria 方法——创建一个新的 Criteria 对象

createCriteria 方法用于创建一个新的 Criteria 对象。该方法用在执行连接查询时。

方法声明

```
public Criteria createCriteria(String associationPath)
                                throws HibernateException
```

入口参数：用于指定映射的联合属性。

返回值：Criteria 对象。

抛出异常：HibernateException。

示例

将主题表与类别表通过 typeid 字段相关联，查询类别为“Java 技术专区”的全部主题信息。

```
Session session=sessionFactory.openSession();
Criteria criteria = session.createCriteria(TbTopic.class);
criteria. createCriteria("tbType").add(Restrictions.eq("typeName","Java 技术专区"));
List list=criteria.list();
```

在上面的代码中，tbType 一定要与 TbTopic 类中对应的属性名相同。

17.6.4 createAlias 方法——建立连接查询

createAlias 方法用于建立连接查询。一般应用于不希望创建新的 criteria 实例时。

方法声明 1

```
public Criteria createAlias(String associationPath,
                            String alias)
                    throws HibernateException
```

入口参数：associationPath 用于指定映射的联合属性；alias 用于指定别名。

返回值：Criteria 对象。

抛出异常：HibernateException。

示例

将主题表与回复表建立内联接的代码如下：

```
Session session=sessionFactory.openSession();
Criteria criteria = session.createCriteria(TbTopic.class)
        .createAlias("tbReplies", "r");
List list=criteria.list();
```

方法声明 2

```
public Criteria createAlias(String associationPath,
                            String alias,
```

```
                                    int joinType)
                                throws HibernateException
```

入口参数：associationPath 用于指定映射的联合属性；alias 用于指定别名；joinType 用于指定连接的类型。

返回值：Criteria 对象。

抛出异常：HibernateException。

示例

将主题表与回复表建立在外联接的代码如下：

```
Session session=sessionFactory.openSession();
Criteria criteria = session.createCriteria(TbTopic.class)
        .createAlias("tbReplies","r",CriteriaSpecification.LEFT_JOIN);
List list=criteria.list();
```

17.6.5 list 方法——返回一个 List 集合

list 方法用来返回一个 List 集合，该集合中保存着与查询条件相匹配的查询结果。

方法声明

```
public List list()
        throws HibernateException
```

返回值：List 集合。

抛出异常：HibernateException。

示例

应用 QBC 查询，查询年龄大于等于 30 的用户信息，并将查询结果保存到 List 集合中。

```
Session session=sessionFactory.openSession();
List list=null;
try{
    Criteria criteria=session.createCriteria(UserForm.class);
    criteria.add(Restrictions.gt("age",30));
    list=criteria.list();
}catch(Exception e){
        System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

17.6.6 setMaxResults 方法——设置最大返回记录数

setMaxResults 方法用于指定返回的结果集中最大的记录条数，该方法通常用在数据分页时。

方法声明

```
public Criteria setMaxResults(int maxResults)
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：用于指定最大的记录条数。

返回值：Criteria 对象。

示例

获取用户信息表中的前 3 条记录。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setMaxResults(3);          //设置最大的记录数
list=criteria.list();
```

17.6.7 setFirsResult 方法——设置结果集的起始点

setFirsResult 方法用于指定返回的结果集的起始点，该方法通常用在数据分页时。

方法声明

```
public Criteria setFirstResult(int firstResult)
```

入口参数：用于指定返回结果集的起始点。

返回值：Criteria 对象。

示例

获取用户信息表中第 3 条记录以后的全部数据，但不包括第 3 条。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setFirstResult(3);
list=criteria.list();
```

17.6.8 uniqueResult 方法——单值检索

uniqueResult 方法用于返回唯一的结果对象。该方法一般用在确信只有一个满足条件的记录时使用，如果结果集的记录条数超过将会抛出异常。

方法声明

```
public Object uniqueResult()
                        throws HibernateException
```

返回值：唯一的结果对象或是 null。

抛出异常：HibernateException。

示例

获取用户信息表中的第一条记录，并保存到 UserForm 对象中。

```
Criteria criteria=session.createCriteria(UserForm.class);
UserForm uf=(UserForm)criteria.setMaxResults(1).uniqueResult();
```

17.6.9 setProjection 方法——设置查询的投影列

setProjection 方法用于设置查询的投影列。该方法与 Projections 类结合不仅可以实现简单的投影查询，而且可以实现数据的分组统计。

方法声明

```
public Criteria setProjection(Projection projection)
```

入口参数：Projection 对象，用于指定投影列或是分组统计条件。

返回值：Criteria 对象。

示例

应用投影查询，查询数据表中的全部用户 ID。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.id());
list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    System.out.println(it.next());
}
```

示例

应用投影查询，查询数据表中的全部用户 ID 和用户名。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.projectionList()
        .add(Projections.id())
        .add(Projections.property("username"))
);
List list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    Object[] o=(Object[])it.next();
    System.out.println("ID:"+o[0]+"用户名："+o[1]);
}
```

示例

获取订单信息表中的合计金额。

```
Criteria criteria=session.createCriteria(OrderForm.class);
criteria.setProjection(Projections.sum("money"));
double sumMoney=( Double)criteria.uniqueResult();
System.out.println("合计金额："+sumMoney);
```

17.7 Projections 类——帮助 criteria 接口完成分组与统计

Projections 类位于 org.hibernate.criterion 包中，用于帮助 criteria 接口完成数据的分组查询和统计功能。

17.7.1 projectionList 方法——创建多个投影列

projectionList 方法用于创建多个投影列，每个投影列由该方法返回的 ProjectionList 对象的 add 方法进行添加。add 方法的参数为 Projections 类的相关

PDF电子书基地 http://dayo1982.400gb.com

方法的返回值。

方法声明

```
public static ProjectionList projectionList()
```

返回值：ProjectionList 类的对象。ProjectionList 类为 Projection 类的子类。

示例

查询数据表中的全部用户名。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.projectionList()
        .add(Projections.property("username")));
list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    System.out.println("用户名："+it.next());
}
```

示例

在应用 projectionList 方法用于创建多个投影列时，可以通过调用多个 add 方法设置多个投影列，例如下面的代码。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.projectionList()
        .add(Projections.property("username"))
        .add(Projections.property("pwd"))
);
List list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    Object[] o=(Object[])it.next();
    System.out.println("用户名："+o[0]+"密码："+o[1]);
}
```

17.7.2 id 方法——投影对象标识符

id 方法用于指明投影对象标识符。

方法声明

```
public static IdentifierProjection id()
```

返回值：IdentifierProjection 类的对象，IdentifierProjection 类为 Projection 类的子类。

示例

查询数据表中的全部用户 ID。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.id());
```

```
list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    System.out.println("用户 ID：  "+it.next());
}
```

17.7.3 property 方法——指明投影对象的某个属性

property 方法用于指明投影对象的某个属性。

方法声明

```
public static PropertyProjection property(String propertyName)
```

返回值：PropertyProjection 类的对象，PropertyProjection 类为 Projection 类的子类。

示例

查询数据表中的全部用户名。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.property("username"));
list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
    System.out.println("用户 ID：  "+it.next());
}
```

17.7.4 distinct 方法——设置查询结果的唯一性

distinct 方法用于设置查询结果的唯一性。

方法声明

```
public static Projection distinct(Projection proj)
```

入口参数：用于指定投影的列，通常为 Projections 对象的 property 方法的返回值。

返回值：Projection 类的对象。

示例

查询数据表中不重复的用户名。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.distinct(Projections.property("username")));
Iterator it =criteria.list().iterator();
while(it.hasNext()){
    System.out.println("用户名：  "+it.next());
}
```

17.7.5 groupProperty 方法——设置参加分组的属性列

groupProperty 方法用于设置参加分组的属性列。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public static PropertyProjection groupProperty(String propertyName)
```

入口参数：用于指定进行分组的属性列的列名。

返回值：PropertyProjection 类的对象。PropertyProjection 类为 Projection 类的子类。

示例

对用户信息表中用户名进行分组，并统计每个用户名的人数。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.projectionList()
.add(Projections.groupProperty("username"))
   .add(Projections.rowCount(),"count")
);
list=criteria.list();
Iterator it=list.iterator();
while(it.hasNext()){
      Object[] o=(Object[])it.next();
      System.out.println("用户名："+o[0]+"人数："+o[1]);
}
```

17.7.6 avg 方法——计算某一列的平均值

avg 方法用于计算某一列的平均值。

方法声明

```
public static AggregateProjection avg(String propertyName)
```

入口参数：用于指定求平均值的属性列的列名。

返回值：AggregateProjection 类的对象，AggregateProjection 类为 Projection 类的子类。

示例

获取成绩信息表中的平均成绩。

```
Criteria criteria=session.createCriteria(ResultForm.class);
criteria.setProjection(Projections.avg("achievement"));
double avg=(Double)criteria.uniqueResult();
System.out.println("平均成绩："+avg);
```

17.7.7 sum 方法——对某一列进行求和

sum 方法用于对某一列进行求和。

方法声明

```
public static AggregateProjection sum(String propertyName)
```

入口参数：用于指定求和的属性列的列名。

返回值：AggregateProjection 类的对象，AggregateProjection 类为 Projection 类的子类。

示例

获取成绩信息表中的总成绩。

```
Criteria criteria=session.createCriteria(ResultForm.class);
criteria.setProjection(Projections.sum("achievement "));
double sum=(Double)criteria.uniqueResult();
System.out.println("总成绩："+sum);
```

17.7.8 max 方法——计算某一列的最大值

max 方法用于计算某一列的最大值。

方法声明

```
public static AggregateProjection max(String propertyName)
```

入口参数：用于指定计算最大值的属性列的列名。

返回值：AggregateProjection 类的对象，AggregateProjection 类为 Projection 类的子类。

示例

获取成绩信息表中的最高成绩。

```
Criteria criteria=session.createCriteria(ResultForm.class);
criteria.setProjection(Projections.max("achievement "));
double max=(Double)criteria.uniqueResult();
System.out.println("最高成绩："+max);
```

17.7.9 min 方法——计算某一列的最小值

min 方法用于计算某一列的最小值。

方法声明

```
public static AggregateProjection max(String propertyName)
```

入口参数：用于指定计算最小值的属性列的列名。

返回值：AggregateProjection 类的对象，AggregateProjection 类为 Projection 类的子类。

示例

获取成绩信息表中的最低成绩。

```
Criteria criteria=session.createCriteria(ResultForm.class);
criteria.setProjection(Projections.max("achievement "));
double min=(Double)criteria.uniqueResult();
System.out.println("最低成绩："+min);
```


PDF电子书基地 http://dayo1982.400gb.com

17.7.10 rowCount 方法——满足条件的记录的数目

rowCount 方法用于返回满足条件的记录的数目，相当于 SQL 中的 count(*)。

方法声明

```
public static Projection rowCount()
```

返回值：Projection 类的对象。

示例

获取用户名为 sk 的记录数。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.eq("username", "sk"));
criteria.setProjection(Projections.rowCount());
int value=(Integer)criteria.uniqueResult();
System.out.println("记录数："+value);
```

17.7.11 count 方法——返回某个属性的数目

count 方法用于返回某个属性的数目。

方法声明

```
public static CountProjection count(String propertyName)
```

入口参数：用于指定计算某个属性数目的属性列的列名。

返回值：CountProjection 类的对象，CountProjection 类是 Projection 类的子类。

示例

获取 id 列的记录数。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.setProjection(Projections.count("id"));
int max=(Integer)criteria.uniqueResult();
System.out.println("记录数："+max);
```

17.8 Restrictions 类——创建查询条件

Restrictions 类位于 org.hibernate.criterion 包中。Restrictions 类提供了大量的静态方法，如 eq、like、lt、gt 和 between 等方法，用于创建查询条件。Restrictions 类是 Hibernate 3.X 中提出用于替代 Hibernate 2.X 中的 Expression 类的。

17.8.1 allEq 方法——设置一系列的相等条件

allEq 方法用于设置一系列的相等条件，相当于多个 Restrictions.eq 方法的叠加。

说明

应用 allEq 方法时，生成的相等表达式是通过 and 来连接的，也就是需要满足添加的所有条件。

方法声明

```
public static Criterion allEq(Map propertyNameValues)
```

入口参数：参数为一个 Map 对象，其中包含了多个属性—值的对应关系。

返回值：Criterion 对象的实例。

示例

查询用户名为 sk，并且密码为 111 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
Map map=new HashMap();
map.put("username", "sk");
map.put("pwd", "111");
criteria.add(Restrictions.allEq(map));
list=criteria.list();
```

17.8.2 eq 方法——设置等于条件

eq 方法用于设置等于条件，相当于 SQL 语句中的“field=value”。

方法声明

```
public static SimpleExpression eq(String propertyName,
                                  Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：SimpleExpression 类的对象，SimpleExpression 类为 Criterion 接口的实现类。

示例

查询用户名等于 wgh 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.eq("username","wgh"));
list=criteria.list();
```

17.8.3 lt 方法——设置小于条件

lt 方法用于设置小于条件，相当于 SQL 语句中的“field<value”。

方法声明

```
public static SimpleExpression lt(String propertyName,
                                  Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：SimpleExpression 类的对象，SimpleExpression 类为 Criterion 接口的实现类。

示例

查询用户名等于 sk，并且年龄小于 20 的用户信息。

PDF电子书基地 http://dayo1982.400gb.com

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.eq("username","sk"))
.add(Restrictions.lt("age",20));
list=criteria.list();
```

17.8.4 le 方法——设置小于等于条件

le 方法用于设置小于等于条件，相当于 SQL 语句中的“field<=value”。

方法声明

```
public static SimpleExpression le(String propertyName,
                                  Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：SimpleExpression 类的对象，SimpleExpression 类为 Criterion 接口的实现类。

示例

查询用户名等于 wgh，并且年龄小于等于 20 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.eq("username","wgh"))
.add(Restrictions.le("age",20));
list=criteria.list();
```

17.8.5 gt 方法——设置大于条件

gt 方法用于设置大于条件，相当于 SQL 语句中的“field>value”。

方法声明

```
public static SimpleExpression gt(String propertyName,
                                  Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：SimpleExpression 类的对象，SimpleExpression 类为 Criterion 接口的实现类。

示例

查询用户 id 大于 100 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.gt("id",100));
list=criteria.list();
```

17.8.6 ge 方法——设置大于等于条件

gt 方法用于设置大于等于条件，相当于 SQL 语句中的“field>=value”。

方法声明

```
public static SimpleExpression ge(String propertyName,
                                  Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：SimpleExpression 类的对象，SimpleExpression 类为 Criterion 接口的实现类。

示例

查询用户 id 大于等于 100 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.ge("id",100));
list=criteria.list();
```

17.8.7 between 方法——设置在两者之间的条件

between 方法用于设置在两者之间的条件，相当于 SQL 的“field between value1 and value2”。

方法声明

```
public static Criterion between(String propertyName,
                                Object lo,
                                Object hi)
```

入口参数：propertyName 用于指定属性名；lo 用于指定最小值；hi 用于指定最大值。

返回值：Criterion 对象的实例。

示例

查询年龄在 20～40 之间（包括 20 和 40）的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.le("age", 20,40));
list=criteria.list();
```

17.8.8 like 方法——设置模糊查询的条件

like 方法用于设置模糊查询的条件，相当于 SQL 的“field like value”。

方法声明

```
public static Criterion ilike(String propertyName,
                              Object value)
```

入口参数：propertyName 用于指定属性名；value 用于指定属性值。

返回值：Criterion 对象的实例。

示例

查询用户名中包括“h”字符的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.like("username","%h%"));
list=criteria.list();
```


PDF电子书基地 http://dayo1982.400gb.com

示例

查询用户名以“w”开头的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.like("username","w%"));
list=criteria.list();
```

17.8.9 in 方法——判断是否在给定的数组内

in 方法用于检查属性值是否在给定的数组内，相当于 SQL 的 in 子句。

方法声明

```
public static Criterion in(String propertyName,
                           Object[] values)
```

入口参数：propertyName 用于指定属性名；values 用于指定属性值，通常为对象数组。

返回值：Criterion 对象的实例。

示例

查询学历为“大专”、“本科”或“研究生”的员工信息。

```
Criteria criteria=session.createCriteria(PersonForm.class);
String[] arr={"大专","本科","研究生"};
criteria.add(Restrictions.in("knowledge",arr));
list=criteria.list();
```

17.8.10 and 方法——逻辑与操作

and 方法用于完成两个表达式的逻辑与操作。

方法声明

```
public static LogicalExpression and(Criterion lhs,
                                    Criterion rhs)
```

入口参数：lhs 用于指定第一个表达式；rhs 用于指定第二个表达式。

返回值：LogicalExpression 类的对象，LogicalExpression 类为 Criterion 接口的实现类。

示例

查询学历为“大专”并且年龄大于 20 的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.and(Restrictions.eq("knowledge","大专"), Restrictions.gt("age",20)));
list=criteria.list();
```

17.8.11 or 方法——逻辑或操作

or 方法用于完成两个表达式的逻辑或操作。

方法声明

```
public static LogicalExpression or(Criterion lhs,
                                   Criterion rhs)
```

入口参数：lhs 用于指定第一个表达式；rhs 用于指定第二个表达式。

返回值：LogicalExpression 类的对象，LogicalExpression 类为 Criterion 接口的实现类。

示例

查询学历为“大专”或者工龄大于 5 的员工信息。

```
Criteria criteria=session.createCriteria(PersonForm.class);
criteria.add(Restrictions.or(Restrictions.eq("knowledge","大专"), Restrictions.g t("work Ag e", 5)));
list=criteria.list();
```

17.8.12 not 方法——逻辑非操作

not 方法用于完成逻辑非操作。

方法声明

```
public static Criterion not(Criterion expression)
```

入口参数：expression 用于指定一个表达式。

返回值：Criterion 对象的实例。

示例

查询学历不为“大专”、“本科”和“研究生”的员工信息。

```
Criteria criteria=session.createCriteria(PersonForm.class);
String[] arr={"大专","本科","研究生"};
criteria.add(Restrictions.not(Restrictions.in("knowledge",arr)));
list=criteria.list();
```

17.8.13 isNotNull 方法——判断是否不为 null

isNotNull 方法用于判断是否不为 null。

方法声明

```
public static Criterion isNotNull(String propertyName)
```

入口参数：propertyName 用于指定要判断是否不为 null 的属性的属性名。

返回值：Criterion 对象的实例。

示例

查询简历不为空的员工信息。

```
Criteria criteria=session.createCriteria(PersonForm.class);
criteria.add(Restrictions. isNotNull ("memo"));
list=criteria.list();
```


PDF电子书基地 http://dayo1982.400gb.com

17.8.14 isNull 方法——判断是否为 null

isNull 方法用于判断是否为 null。

方法声明

```
public static Criterion isNull(String propertyName)
```

入口参数：propertyName 用于指定要判断是否为 null 的属性的属性名。

返回值：Criterion 对象的实例。

示例

查询简历为空的员工信息。

```
Criteria criteria=session.createCriteria(PersonForm.class);
criteria.add(Restrictions. isNull ("memo"));
list=criteria.list();
```

17.8.15 sqlRestriction 方法——通过 SQL 语句限定查询条件

sqlRestriction 方法用于设置直接通过 SQL 语句限定查询条件。

方法声明

```
public static Criterion isNull(String propertyName)
```

入口参数：propertyName 用于指定要判断是否为 null 的属性的属性名。

返回值：Criterion 对象的实例。

示例

查询用户名以“w”开头的用户信息。

```
Criteria criteria=session.createCriteria(UserForm.class);
criteria.add(Restrictions.sqlRestriction("{alias}.username like (?)","w%",Hibernate.STRING));
list=criteria.list();
```

示例

查询 2009-1-1～2009-6-30 之间的订单信息。

```
Criteria criteria=session.createCriteria(OrderForm.class);
Date[] time={new Date(new java.util.GregorianCalendar(2009,1,1).getTimeInMillis()),new Dat
e(new java.util.GregorianCalendar(2009,6,30).getTimeInMillis())};
Type[] types={Hibernate.DATE,Hibernate.DATE};
criteria.add(Restrictions.sqlRestriction("{alias}.orderDate between(?) and (?)",time,types) );
list=criteria.list();
```

17.9 SQLQuery 接口——SQL 查询接口

SQLQuery 接口位于 org.hibernate 包中。SQLQuery 接口是 Query 接口的子接口，用于完成对原生 SQL 查询执行控制。

17.9.1 addEntity 方法——将别名和实体类联系

addEntity 方法用于将 SQL 表的别名和实体类联系起来，并且确定查询结果集的形态。

方法声明 1

```
public SQLQuery addEntity(Class entityClass)
```

入口参数：用于指定实体类。

返回值：SQLQuery 对象的实例。

示例

通过本地 SQL 查询，查询全部的用户信息。

```
Session session=sessionFactory.openSession();
String sql="select * from tb_user";
List list=null;
List rtnList=new ArrayList();
try{
    SQLQuery query=session.createSQLQuery(sql);
    query.addEntity(UserForm.class);                    //绑定实体类
    list=query.list();
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

在上面的代码中，如果不应用 addEntity 方法也可实现 SQL 查询，但返回的结果集对象为 Object 数组。

方法声明 2

```
public SQLQuery addEntity(String alias,
                          Class entityClass)
```

入口参数：alias 用于为实体类指定别名；entityClass 用于指定实体类。

返回值：SQLQuery 对象的实例。

示例

通过本地 SQL 查询，查询用户名为 wgh 用户信息，此时指定实体类的别名为 u。

```
Session session=sessionFactory.openSession();
String sql="select {u.*} from tb_user u where u.username='wgh'";
List list=null;
List rtnList=new ArrayList();
try{
    SQLQuery query=session.createSQLQuery(sql);
    query.addEntity("u",UserForm.class);
    list=query.list();
```


PDF电子书基地 http://dayo1982.400gb.com

```
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

方法声明 3

```
public SQLQuery addEntity(String alias,
                          Class entityClass,
                          LockMode lockMode)
```

入口参数：alias 用于为实体类指定别名；entityClass 用于指定实体类；lockMode 用于指定锁定模式。

LockMode 类提供的几种锁定模式

锁 定 模 式	功 能 简 介
LockMode.NONE	表示无锁机制，是 Hibernate 的默认值。采用该种方式首先在 Hibernate 缓存中检索，存在则直接返回对象的引用；否则到数据库中加载对象。是 Hibernate 的默认值
LockMode.READ	无论 Hibernate 缓存中是否存在，总是直接到数据库中加载。如果在映射文件中设置了版本元素，则比较缓存中对象的版本与数据库中记录的版本是否一致
LockMode.UPGRADE	在 LockMode.READ 模式的基础上，如果数据库系统支持悲观锁（如 MySQL 和 Oracle），则执行 select…for update 语句，否则执行普通的 select 语句
LockMode.UPGRADE_NOWAIT	在上一模式的基础上，如果是 Oracle 数据库，则执行 select…for update nowait 语句，“nowait”的作用是如果执行该 select 语句的事务不能立刻获得悲观锁，会立刻抛出锁定异常，而不会去等待其他事务释放锁定

返回值：SQLQuery 对象的实例。

示例

通过本地 SQL 查询，查询用户名为 sk 的用户信息，此时指定实体类的别名为 u，并且指定锁定模式为 LockMode.READ。

```
Session session=sessionFactory.openSession();
String sql="select {u.*} from tb_user u where u.username='sk'";
List list=null;
List rtnList=new ArrayList();
try{
    SQLQuery query=session.createSQLQuery(sql);
    query.addEntity("u",UserForm.class,LockMode.READ);
    list=query.list();
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

方法声明 4

```
public SQLQuery addEntity(String entityName)
```

入口参数：entityName 为表示实体类的字符串（这里需要指定对应的包）。

返回值：SQLQuery 对象的实例。

示例

通过本地 SQL 查询，查询用户名为 sk 的用户信息。

```
Session session=sessionFactory.openSession();
String sql="select * from tb_user where username='sk'";
List list=null;
List rtnList=new ArrayList();
try{
    SQLQuery query=session.createSQLQuery(sql);
    query.addEntity("com.model.UserForm");                    //绑定实体类
    list=query.list();
}catch(Exception e){
    System.out.println("查询用户信息时的错误信息："+e.getMessage());
}
```

方法声明 5

```
public SQLQuery addEntity(String alias,
                          String entityName)
```

入口参数：alias 用于为实体类指定别名；entityName 为表示实体类的字符串（这里需要指定对应的包）。

返回值：SQLQuery 对象的实例。

示例

通过本地 SQL 查询，查询用户名为 wgh 的用户信息。

```
Session session=sessionFactory.openSession();
String sql="select {u.*} from tb_user u where u.username='wgh'";
SQLQuery query=session.createSQLQuery(sql);
query.addEntity("u","com.model.UserForm");
List list=query.list();
```

17.9.2 addJoin 方法——实体和集合相关联

addJoin 方法用于载入其他的实体和集合相关联。

方法声明

```
public SQLQuery addJoin(String alias,
                        String path)
```

入口参数：alias 用于为关联的实体类指定别名；path 为表示相关联实体类的字符串（这里需要指定对应的包）。

返回值：SQLQuery 对象的实例。

PDF电子书基地 http://dayo1982.400gb.com

示例

应用本地 SQL 查询，通过关联从订单主表和订单明细表中查询订单的详细信息，订单主表和订单明细表之间通过订单 ID 相关联。

```
Session session=sessionFactory.openSession();
String sql="select {order.*}, {order_detail.*} from tb_order order,tb_orderDetail order_detail
where order.id = order_detail.orderId";
    List list=null;
    List rtnList=new ArrayList();
    try{
        SQLQuery query=session.createSQLQuery(sql);
        query.addEntity("order", Order.class);
        query.addJoin("order_detail", "com.model.OrderDetailForm");
        list=query.list();
    }catch(Exception e){
        System.out.println("查询用户信息时的错误信息："+e.getMessage());
    }
```

17.9.3 addScalar 方法——返回标量和实体的结合体

addScalar 方法用于返回一个简单的标量值或者一个标量和实体的结合体。

方法声明

```
public SQLQuery addScalar(String columnAlias,
                          Type type)
```

入口参数：columnAlias 用于指定列的别名；type 用于指定列的数据类型。

返回值：SQLQuery 对象的实例。

示例

通过 SQL 从学生成绩信息表中获取英语科目的平均成绩。

```
String sql="select avg(result.english) as avgEnglish from tb_result result";
SQLQuery query= session.createSQLQuery(sql)
.addScalar("avgEnglish", Hibernate.FLOAT);
Float avgEnglish = (Float)query.uniqueResult();
```

第 18 章　Hibernate 配置属性

18.1　dialect 属性——数据库方言

hibernate.dialect 属性用于指定连接数据库使用的 SQL 语言。

语法 1

采用 Java 属性文件格式的配置文件时，指定 dialect 属性的语法如下。

```
hibernate.dialect= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 dialect 属性的语法如下。

```
<property name="hibernate.dialect">
属性值
</property>
```

属性值

hibernate.dialect 属性的属性值为 Hibernate 所支持的 SQL 方言。Hibernate 支持的 SQL 方言如下表所示。

Hibernate 支持的 SQL 方言

数　据　库	Hibernate 方言类
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle（any version）	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect

PDF电子书基地 http://dayo1982.400gb.com

续表

数 据 库	Hibernate 方言类
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

示例

采用XML格式的配置文件连接Microsoft SQL Server数据库时设置数据库方言。

```
<property name="hibernate.dialect">
org.hibernate.dialect.SQLServerDialect
</property>
```

采用 Java 属性文件格式的配置文件连接 Microsoft SQL Server 数据库时设置数据库方言。

```
hibernate.dialect= org.hibernate.dialect.SQLServerDialect
```

示例

采用 XML 格式的配置文件连接 MySQL 数据库时设置数据库方言。

```
<property name="hibernate.dialect">
org.hibernate.dialect.MySQLDialect
</property>
```

采用 Java 属性文件格式的配置文件连接 MySQL 数据库时设置数据库方言。

```
hibernate.dialect= org.hibernate.dialect.MySQLDialect
```

示例

采用 XML 格式的配置文件连接 Oracle 9i/10g 数据库时设置数据库方言。

```
<property name="hibernate.dialect">
org.hibernate.dialect.Oracle9Dialect
</property>
```

采用Java属性文件格式的配置文件连接Oracle 9i/10g数据库时设置数据库方言。

```
hibernate.dialect= org.hibernate.dialect.Oracle9Dialect
```

18.2 connection.driver_class 属性——数据库驱动

hibernate.connection.driver_class 属性用于指定连接数据库使用的驱动程序。

语法 1

采用 Java 属性文件格式的配置文件时，指定 connection.driver_class 属性的语法如下。

```
hibernate.connection.driver_class = 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 connection.driver_class 属性的语法如下。

```
<property name="hibernate.connection.driver_class ">
属性值
</property>
```

属性值

hibernate.connection.driver_class 属性的属性值为各数据库所提供的驱动类。常用的数据库驱动类如下表所示。

常用的数据库驱动类

数 据 库	驱 动 类
DB2	COM.ibm.db2.jdbc.app.DB2Driver
PostgreSQL	org.postgresql.Driver
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
Sybase	com.sybase.jdbc2.jdbc.SybDriver
Microsoft SQL Server	com.microsoft.jdbc.sqlserver.SQLServerDriver
SAP DB	com.sap.dbtech.jdbc.DriverSapDB
Informix	com.Informix.jdbc.IfxDriver
HypersonicSQL	org.hsqldb.jdbcDriver
Mckoi SQL	com.mckoi.JDBCDriver
Interbase	interbase.interclient.Driver
Pointbase	com.pointbase.jdbc.jdbcUniversalDriver

示例

采用 XML 格式的配置文件连接 Microsoft SQL Server 数据库时指定数据库驱动类。

```
<property name="hibernate.connection.driver_class">
com.microsoft.jdbc.sqlserver.SQLServerDriver
</property>
```

采用 Java 属性文件格式的配置文件连接 Microsoft SQL Server 数据库时指定数据库驱动类。

示例

```
hibernate.connection.driver_class=com.microsoft.jdbc.sqlserver.SQLServerDriver
```

示例

采用 XML 格式的配置文件连接 MySQL 数据库时指定数据库驱动类。

```
<property name="hibernate.connection.driver_class">
com.mysql.jdbc.Driver
</property>
```


PDF电子书基地 http://dayo1982.400gb.com

采用 Java 属性文件格式的配置文件连接 MySQL 数据库时指定数据库驱动类。

```
hibernate.connection.driver_class = com.mysql.jdbc.Driver
```

示例

采用 XML 格式的配置文件连接 Oracle 数据库时指定数据库驱动类。

```
<property name="hibernate.connection.driver_class">
oracle.jdbc.driver.OracleDriver
</property>
```

采用 Java 属性文件格式的配置文件连接 Oracle 数据库时指定数据库驱动类。

```
hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver
```

18.3 connection.url 属性——数据库连接 URL

hibernate.connection.url 属性用于指定连接数据库的 URL。

语法 1

采用 Java 属性文件格式的配置文件时，指定 connection.url 属性的语法如下。

```
hibernate.connection.url= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 connection.url 属性的语法如下。

```
<property name="hibernate.connection.url">
属性值
</property>
```

属性值

hibernate.connection.url 属性的属性值为连接指定数据库的 URL 地址。常用数据库的连接时使用的 URL 如下表所示。

常用数据库的连接时使用的 URL

数 据 库	驱 动 类
DB2	jdbc:db2://localhost:5000/db_myDB，其中 db_myDB 为数据库名
PostgreSQL	jdbc:postgresql://localhost/myDB
MySQL	jdbc:mysql://localhost:3306/db_myDB，其中 db_myDB 为数据库名
Oracle	jdbc:oracle:thin:@localhost:1521: db_myDB，其中 db_myDB 为数据库的 SID
Sybase	jdbc:sybase:Tds:localhost:5007/ db_myDB，其中 db_myDB 为数据库名
Microsoft SQL Server	jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_myDB，其中 db_myDB 为数据库名
Informix	jdbc:informix-sqli://123.45.67.89:1533/ db_myDB:INFORMIXSERVER=myserver; user=testuser;password=testpassword，其中 db_myDB 为数据库名

示例

采用 XML 格式的配置文件连接 Microsoft SQL Server 数据库时指定数据库 URL。

```
<property name="hibernate.connection.url">
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_myDB
</property>
```

采用 Java 属性文件格式的配置文件连接 Microsoft SQL Server 数据库时指定数据库 URL。

```
hibernate.connection.url=jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_myDB
```

示例

采用 XML 格式的配置文件连接 MySQL 数据库时指定数据库的 URL。

```
<property name="hibernate.connection.url">
jdbc:mysql://localhost:3306/db_myDB
</property>
```

采用 Java 属性文件格式的配置文件连接 MySQL 数据库时指定数据库的 URL。

```
hibernate.connection.url= jdbc:mysql://localhost:3306/db_myDB
```

18.4 connection.username 属性——数据库连接用户名

hibernate.connection.username 属性用于指定连接数据库时所需的用户名。

语法 1

采用 Java 属性文件格式的配置文件时，指定 connection.username 属性的语法如下。

```
hibernate.connection.username= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 connection.username 属性的语法如下。

```
<property name="hibernate.connection.username">
属性值
</property>
```

属性值

hibernate.connection.username 属性的属性值为连接数据库时所需的用户名。

示例

采用XML格式的配置文件连接Microsoft SQL Server数据库时指定数据库的登录用户为 sa。

```
<property name=" hibernate.connection.username ">
sa
</property>
```


PDF电子书基地 http://dayo1982.400gb.com

采用 Java 属性文件格式的配置文件连接 Microsoft SQL Server 数据库时指定数据库的登录用户为 sa。

```
hibernate.connection.username=sa
```

示例

采用 XML 格式的配置文件连接 MySQL 数据库时指定数据库的登录用户为 root。

```
<property name=" hibernate.connection.username ">
root
</property>
```

采用 Java 属性文件格式的配置文件连接 MySQL 数据库时指定数据库的登录用户为 root。

```
hibernate.connection.username=root
```

18.5 connection.password 属性——数据库连接密码

hibernate.connection.password 属性用于指定连接数据库时所需用户名的密码。

语法 1

采用 Java 属性文件格式的配置文件时，指定 connection.password 属性的语法如下。

```
hibernate.connection.password=属性值
```

语法 2

采用 XML 格式的配置文件时，指定 connection.password 属性的语法如下。

```
<property name="hibernate.connection.password">
属性值
</property>
```

属性值

hibernate.connection.password 属性的属性值为连接数据库时所需的用户名。

示例

采用 XML 格式的配置文件连接数据库时，指定数据库的登录用户的密码为空。

```
<property name=" hibernate.connection.password "></property>
```

采用 Java 属性文件格式的配置文件连接数据库时，指定数据库的登录用户的密码为空。

```
hibernate.connection.password=
```

示例

采用 XML 格式的配置文件连接数据库时，指定数据库的登录用户的密码为 111。

```
<property name=" hibernate.connection.password ">
111
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，指定数据库的登录用户的密码为 111。

```
hibernate.connection.password=111
```

18.6 show_sql 属性——显示 SQL 语句

hibernate.show_sql 属性用于指定是否在控制台上输出 SQL 语句。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.show_sql 属性的语法如下。

```
hibernate.show_sql= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.show_sql 属性的语法如下。

```
<property name=" hibernate.show_sql ">
属性值
</property>
```

属性值

hibernate.show_sql 属性的属性值为 true 或 false。其中，为 true 表示在控制台上输出 SQL 语句，为 false 表示不在控制台上输出 SQL 语句，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，指定在控制台上输出 SQL 语句。

```
<property name=" hibernate.show_sql ">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，指定在控制台上输出 SQL 语句。

```
hibernate.show_sql=true
```

18.7 format_sql 属性——格式化 SQL 语句

hibernate.format_sql 属性用于指定是否按照标准格式在控制台上输出 SQL 语句。该属性只有当 show_sql 属性值为 true 时才有效。建议在程序调试后将其删除。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.format_sql 属性的语法如下。

```
hibernate.format_sql= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.format_sql 属性的语法如下。

PDF电子书基地 http://dayo1982.400gb.com

```
<property name=" hibernate.format_sql ">
属性值
</property>
```

属性值

hibernate.format_sql 属性的属性值为 true 或 false。其中，为 true 表示采用标准格式在控制台上输出 SQL 语句，为 false 表示不采用标准格式在控制台上输出 SQL 语句，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，指定采用标准格式在控制台上输出 SQL 语句。

```
<property name=" hibernate.format_sql ">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，指定采用标准格式在控制台上输出 SQL 语句。

```
hibernate.format_sql=true
```

18.8 use_sql_comments 属性——输出注释信息

hibernate.use_sql_comments 属性用于指定是否在 SQL 语句中输出便于调试的注释信息。该属性只有当 show_sql 属性值为 true 时才有效。建议在程序调试后将其删除。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.use_sql_comments 属性的语法如下。

```
hibernate.use_sql_comments= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.use_sql_comments 属性的语法如下。

```
<property name=" hibernate.use_sql_comments ">
属性值
</property>
```

属性值

hibernate.use_sql_comments 属性的属性值为 true 或 false。其中，为 true 表示在 SQL 语句中输出便于调试的注释信息，为 false 表示不在 SQL 语句中输出便于调试的注释信息，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，指定在 SQL 语句中输出便于调试的注释信息。

```
<property name=" hibernate.use_sql_comments ">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，指定在 SQL 语句中输出便于调试的注释信息。

```
hibernate.use_sql_comments=true
```

18.9 default_schema 属性——默认的表空间

hibernate.default_schema 属性用于在生成的 SQL 语句中，使用指定的 schema/tablespace 限定未限定的表名。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.default_schema 属性的语法如下。

```
hibernate.default_schema= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.default_schema 属性的语法如下。

```
<property name="hibernate.default_schema">
属性值
</property>
```

属性值

hibernate.default_schema 属性的属性值为指定数据库的表空间或架构。

示例

在连接 Oracle 数据库时，指定默认的表空间为 mrsoft。采用 XML 格式的配置文件连接数据库时的代码如下。

```
<property name="hibernate.default_schema">
mrsoft
</property>
```

在连接 Oracle 数据库时，指定默认的表空间为 mrsoft。采用 Java 属性文件格式的配置文件连接数据库时的代码如下。

```
hibernate.default_schema=mrsoft
```

18.10 default_catalog 属性——默认的表目录

hibernate.default_catalog 属性用于在生成的 SQL 语句中，使用指定的 catalog

PDF电子书基地 http://dayo1982.400gb.com

限定未限定的表名。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.default_catalog 属性的语法如下。

```
hibernate.default_catalog= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.default_catalog 属性的语法如下。

```
<property name="hibernate.default_catalog">
属性值
</property>
```

属性值

hibernate.default_catalog 属性的属性值为指定 OLE DB 数据源中包含该对象的目录。

示例

在连接 SQL Server 数据库时，指定默认的表目录为 dbo。采用 XML 格式的配置文件连接数据库时的代码如下。

```
<property name="hibernate.default_catalog">
dbo
</property>
```

在连接 SQL Server 数据库时，指定默认的表目录为 dbo。采用 Java 属性文件格式的配置文件连接数据库时的代码如下。

```
hibernate.default_catalog=dbo
```

18.11 session_factory_name 属性——绑定到 JNDI

hibernate.session_factory_name 属性用于 SessionFactory 创建后，将自动使用这个名字绑定到 JNDI 中。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.session_factory_name 属性的语法如下。

```
hibernate.session_factory_name= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.session_factory_name 属性的语法如下。

```
<property name="hibernate.session_factory_name">
属性值
</property>
```

属性值

hibernate.session_factory_name 属性的属性值为绑定到 JNDI 上的 SessionFactory 对象名。

示例

采用 XML 格式的配置文件连接数据库时，在 SessionFactory 创建后，自动将 SessionFactory 对象 hibernate/sessionFactory 绑定到 JNDI 上。

```
<property name="hibernate.session_factory_name">
hibernate/sessionFactory
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，在 SessionFactory 创建后，自动将 SessionFactory 对象 hibernate/sessionFactory 绑定到 JNDI 上。

```
hibernate.session_factory_name=hibernate/sessionFactory
```

18.12 max_fetch_depth 属性——设置抓取深度

hibernate.max_fetch_depth 属性用于为单向关联（一对一或多对一）的外连接抓取（Outer Join Fetch）树设置最大深度。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.max_fetch_depth 属性的语法如下。

```
hibernate.max_fetch_depth= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.max_fetch_depth 属性的语法如下。

```
<property name="hibernate.max_fetch_depth">
属性值
</property>
```

属性值

hibernate.max_fetch_depth 属性的属性值为单向关联的外连接抓取树的最大深度。值为 0 意味着关闭默认的外连接抓取。建议在 0～3 之间取值。

示例

采用 XML 格式的配置文件连接数据库时，设置单向关联的外连接抓取树的最大深度为 1。

```
<property name="hibernate.max_fetch_depth">
1
</property>
```


PDF电子书基地 http://dayo1982.400gb.com

采用 Java 属性文件格式的配置文件连接数据库时，设置单向关联的外连接抓取树的最大深度为 1。

```
hibernate.max_fetch_depth=1
```

18.13 default_batch_fetch_size 属性——设置默认抓取数量

hibernate.default_batch_fetch_size 属性用于为 Hibernate 关联的批量抓取设置默认数量。

语法 1

采用Java属性文件格式的配置文件时，指定hibernate.default_batch_fetch_size属性的语法如下。

```
hibernate.default_batch_fetch_size= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.default_batch_fetch_size 属性的语法如下。

```
<property name="hibernate.default_batch_fetch_size">
属性值
</property>
```

属性值

hibernate.default_batch_fetch_size 属性的属性值为 Hibernate 关联的批量抓取的默认数量。建议使用 4、8 或 16。

示例

采用 XML 格式的配置文件连接数据库时，设置 Hibernate 关联的批量抓取的默认数量为 4。

```
<property name="hibernate.default_batch_fetch_size">
4
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置 Hibernate 关联的批量抓取的默认数量为 4。

```
hibernate.default_batch_fetch_size=4
```

18.14 default_entity_mode 属性——默认实体表现模式

hibernate.default_entity_mode 属性用于为由 SessionFactory 打开的所有 Session 指定默认的实体表现模式。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.default_entity_mode

属性的语法如下。

```
hibernate.default_entity_mode= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.default_entity_mode 属性的语法如下。

```
<property name="hibernate.default_entity_mode">
属性值
</property>
```

属性值

hibernate.default_entity_mode 属性的属性值为 dynamic-map、dom4j 或 pojo。

示例

采用 XML 格式的配置文件连接数据库时，设置由 SessionFactory 打开的所有 Session 默认的实体表现模式为 pojo。

```
<property name="hibernate.default_entity_mode">
pojo
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置由 SessionFactory 打开的所有 Session 默认的实体表现模式为 pojo。

```
hibernate.default_entity_mode=pojo
```

18.15 order_updates 属性——为 SQL 更新排序

hibernate.order_updates 属性用于设置是否强制 Hibernate 按照被更新数据的主键为 SQL 更新排序。设置为是，将减少在高并发系统中事务的死锁。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.order_updates 属性的语法如下。

```
hibernate.order_updates= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.order_updates 属性的语法如下。

```
<property name="hibernate.order_updates">
属性值
</property>
```

属性值

hibernate.order_updates 属性的属性值为 true 或是 false，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，设置强制 Hibernate 按照被更新

PDF电子书基地 http://dayo1982.400gb.com

数据的主键为 SQL 更新排序。

```
<property name="hibernate.order_updates">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置强制 Hibernate 按照被更新数据的主键为 SQL 更新排序。

```
hibernate.order_updates=true
```

18.16 generate_statistics 属性——收集统计数据

hibernate.generate_statistics 属性用于指定 Hibernate 是否收集有助于性能调节的统计数据。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.generate_statistics 属性的语法如下。

```
hibernate.generate_statistics= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.generate_statistics 属性的语法如下。

```
<property name="hibernate.generate_statistics">
属性值
</property>
```

属性值

hibernate.generate_statistics 属性的属性值为 true 或是 false，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，设置 Hibernate 收集有助于性能调节的统计数据。

```
<property name="hibernate.generate_statistics">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置 Hibernate 收集有助于性能调节的统计数据。

```
hibernate.generate_statistics=true
```

18.17 use_identifer_rollback 属性——重设默认值

hibernate.use_identifer_rollback 属性用于指定在对象被删除时，生成的标识属性是否被重设为默认值。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.use_identifer_rollback 属性的语法如下。

```
hibernate.use_identifer_rollback= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.use_identifer_rollback 属性的语法如下。

```
<property name="hibernate.use_identifer_rollback">
属性值
</property>
```

属性值

hibernate.use_identifer_rollback 属性的属性值为 true 或是 false，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，设置在对象被删除时，生成的标识属性自动重设为默认值。

```
<property name="hibernate.use_identifer_rollback">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置在对象被删除时，生成的标识属性自动重设为默认值。

```
hibernate.use_identifer_rollback=true
```

18.18 cache.use_query_cache 属性——判断是否缓存查询结果

hibernate.cache.use_query_cache 属性用于指定是否缓存查询结果。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.cache.use_query_cache 属性的语法如下。

```
hibernate.cache.use_query_cache= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.cache.use_query_cache 属性的语法如下。

```
<property name="hibernate.cache.use_query_cache">
属性值
</property>
```


PDF电子书基地 http://dayo1982.400gb.com



属性值

hibernate.cache.use_query_cache 属性的属性值为 true 或是 false，默认值为 false。

示例

采用 XML 格式的配置文件连接数据库时，设置缓存查询结果。

```
<property name="hibernate.cache.use_query_cache">
true
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置缓存查询结果。

```
hibernate.cache.use_query_cache=true
```

18.19 cache.use_second_level_cache 属性——判断是否关闭二级缓存

hibernate.cache.use_second_level_cache 属性用于指定是否彻底关闭所有二级缓存。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.cache.use_sec ond _ level_cache 属性的语法如下。

```
hibernate.cache.use_second_level_cache= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.cache.use_second_level_cache 属性的语法如下。

```
<property name="hibernate.cache.use_second_level_cache">
属性值
</property>
```

属性值

hibernate.cache.use_second_level_cache 属性的属性值为 true 或是 false。其中，为 true 时表示启用二级缓存，为 false 时表示彻底关闭所有二级缓存。

示例

采用 XML 格式的配置文件连接数据库时，设置彻底关闭所有二级缓存。

```
<property name="hibernate.cache.use_second_level_cache">
false
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置彻底关闭所有二级缓存。

```
hibernate.cache.use_second_level_cache=false
```

18.20 cache.provider_class 属性——设置二级缓存实现类的全名

hibernate.cache.provider_class 属性用于设置二级缓存实现类的全名。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.cache.provider_class 属性的语法如下。

```
hibernate.cache.provider_class= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.cache.provider_class 属性的语法如下。

```
<property name="hibernate.cache.provider_class">
属性值
</property>
```

属性值

hibernate.cache.provider_class 属性的属性值为 org.hibernate.cache.HashtableCaheProvide（Hashtable）、org.hibernate.cache.EhCacheProvider（EHCache）、org.hibernate.cache.OSCacheProvider（OSCache）、org.hibernate.cache.SwarmCacheProvider（SwarmCache）和 org.hibernate. cache.Tree CacheProvider（JBoss TreeCache）等。

示例

采用 XML 格式的配置文件连接数据库时，设置二级缓存实现类的全名为 OSCache。

```
<property name="hibernate.cache.provider_class">
org.hibernate.cache.OSCacheProvider
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置二级缓存实现类的全名 OSCache。

```
hibernate.cache.provider_class= org.hibernate.cache.OSCacheProvider
```

18.21 cache.region_prefix 属性——设置二级缓存的前缀名称

hibernate.cache.region_prefix 属性用于设置二级缓存的前缀名称。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.cache.region_prefix 属性的语法如下。

```
hibernate.cache.region_prefix= 属性值
```


PDF电子书基地 http://dayo1982.400gb.com



语法 2

采用 XML 格式的配置文件时，指定 hibernate.cache.region_prefix 属性的语法如下。

```
<property name="hibernate.cache.region_prefix">
属性值
</property>
```

属性值

hibernate.cache.region_prefix 属性的属性值为二级缓存的前缀名称。

示例

采用 XML 格式的配置文件连接数据库时，设置二级缓存的前缀名称为 c_mr。

```
<property name="hibernate.cache.region_prefix">
c_mr
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置二级缓存的前缀名称为 c_mr。

```
hibernate.cache.region_prefix=c_mr
```

18.22 cache.use_minimal_puts 属性——判断是否优化二级缓存

hibernate.cache.use_minimal_puts 属性用于设置是否优化二级缓存来最小化读写操作，集群时的缓存优化。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.cache.use_minimal_puts 属性的语法如下。

```
hibernate.cache.use_minimal_puts= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.cache.use_minimal_puts 属性的语法如下。

```
<property name="hibernate.cache.use_minimal_puts">
属性值
</property>
```

属性值

hibernate.cache.use_minimal_puts 属性的属性值为 true 或 false。其中，为 true 表示启用最小化操作，为 false 表示禁用最小化操作，默认值为 true。

示例

采用 XML 格式的配置文件连接数据库时，设置禁用最小化操作。

```
<property name="hibernate.cache.use_minimal_puts">
false
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置禁用最小化操作。

```
hibernate.cache.use_minimal_puts=false
```

18.23 transaction.factory_class 属性——设置事务管理类型

hibernate.transaction.factory_class 属性用于设置事务管理类型。

语法 1

采用 Java 属性文件格式的配置文件时，指定 hibernate.transaction.factory_class 属性的语法如下。

```
hibernate.transaction.factory_class= 属性值
```

语法 2

采用 XML 格式的配置文件时，指定 hibernate.transaction.factory_class 属性的语法如下。

```
<property name="hibernate.transaction.factory_class">
属性值
</property>
```

属性值

hibernate.transaction.factory_class 属性的属性值为包括以下几个，默认使用 org.hibernate.transaction.JDBCTransactionFactory。

- org.hibernate.transaction.JDBCTransactionFactory

请求将会传递给数据库事务服务。默认情况下，Hibernate 使用这种配置。

- org.hibernate.transaction.JTATransactionFactory

请求将会传递给容器管理的 JTA 事务。

- org.hibernate.transaction.CMTTransactionFactory

如果在当前上下文环境（context）中存在一个容器管理的事务，请求将会传递给这个事务，不然将会创建一个新的事务用于处理请求。

示例

采用 XML 格式的配置文件连接数据库时，设置事务管理类型为 JTATransaction Factory。

```
<property name="hibernate.transaction.factory_class">
org.hibernate.transaction.JTATransactionFactory
</property>
```

采用 Java 属性文件格式的配置文件连接数据库时，设置事务管理类型为 JTATransactionFactory。

```
hibernate.transaction.factory_class= org.hibernate.transaction.JTATransactionFactory
```


PDF电子书基地 http://dayo1982.400gb.com

第 19 章　Hibernate 映射节点

19.1　<hibernate-mapping>节点——实体映射

<hibernate-mapping>节点用于定义当前配置文件中映射关系的基本属性。在该节点中定义的属性对这个映射文件中所有的子节点都发挥作用。

语法

```
<hibernate-mapping
        schema="schemaName"
        catalog="catalogName"
        default-cascade="cascade_style"
        default-access="field|property|ClassName"
        default-lazy="true|false"
        auto-import="true|false"
        package="package.name"
 >
</hibernate-mapping>
```

属性

<hibernate-mapping>节点属性说明如下表所示。

<hibernate-mapping>节点属性说明

属　性	说　明
schema	指定数据库 schema 名称
catalog	指定数据库 catalog 名称
default-cascade	用于指定本映射文件使用的默认级联模式。该属性的属性值包括 none（在保存、修改或删除对象时，不对与之关联的对象进行任何操作）、save-update（在保存或修改对象时，级联保存或更新所有与之关联的对象）、delete（当删除本对象时，级联删除所有与之关联的对象）、all（包括 save-update 和 delete 的行为），默认值为 none
default-access	设置默认的属性访问模式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类），多数情况下使用 property
default-lazy	设置默认的是否采用延迟加载模式。延迟加载是指在用到时，才会获取被关联的对象，在非延迟加载的情况下，会自动获取被关联的对象。采用延迟加载在某种情况下，会减少无谓的性能开销

续表

属　性	说　明
auto-import	设置在 HQL 查询中是否可使用不包含包名的类名。如果该属性设置为 true，则 HQL 语句中只需要包括类名即可（from UserForm），Hibernate 将自动根据类名进行匹配，否则 HQL 语句中必须包含全路径名（例如，from com.model.UserForm）。需要注意的是，如果两个不同的包中存在同名类，则必须将该属性设置为 false
package	用于指定本映射文件中映射类的默认包

示例

设置本映射文件中映射类的默认包为 com.model，并且在 HQL 查询时，使用不包含包类的类名。

```
<hibernate-mapping
        auto-import="true"
        package="com.model"
 >
</hibernate-mapping>
```

示例

设置本映射文件使用的属性访问模式为 property，级联模式为 delete。

```
<hibernate-mapping
        default-access="property"
        default-cascade="delete"
>
</hibernate-mapping>
```

19.2　<class>节点——定义类/表的映射关系

<class>节点用于定义类/表的映射关系。

语法

```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
        select-before-update="true|false"
        polymorphism="implicit|explicit"
        where="arbitrary sql where condition"
        persister="PersisterClass"
        batch-size="N"
        optimistic-lock="none|version|dirty|all"
```


PDF电子书基地 http://dayo1982.400gb.com

```
            lazy="true|false"
            entity-name="EntityName"
            check="arbitrary sql check condition"
            rowid="rowid"
            subselect="SQL expression"
            abstract="true|false"
    >
    </class>
```

属性

<class>节点属性说明如下表所示。

<class>节点属性说明

属　性	说　明
name	用于指定类名。如果在<hibernate-mapping>节点中设置了 package 属性，那么这里可以直接使用类名，否则应该使用类全名
table	用于指定类对应的数据表
discriminator-value	用于指定子类识别标识。在多态行为时用于区分不同的子类的值
mutable	用于指定该类的实例是否是可以改变的。当一个表中的记录不会发生变化时，可以将其设置为 false，Hibernabe 在进行脏数据检查时，将忽略此类型数据，从而在一定程度上提升了性能。默认值为 true
schema	用于指定数据库的 schema。使用该属性时，将忽略<hibernate-mapping>节点中设置的 schema 属性的值
catalog	用于指定数据库的 catalog。使用该属性时，将忽略<hibernate-mapping>节点中设置的 catelog 属性的值
proxy	用于指定延迟加载代理类
dynamic-update	用于指定 Update SQL 语句是否动态生成。如果这个属性的值设置为 true，那么生成的 SQL 语句中，只包括那些发生变动的字段。默认值为 false
dynamic-insert	用于指定 Insert SQL 语句是否动态生成。如果这个属性的值设置为 true，那么生成的 SQL 语句中，只包括非空字段。默认值为 false
select-before-update	用于指定在执行 Update 操作时，是否先执行一次查询，确定数据发生变化后，才进行更新
polymorphism	用于设置多态模式
where	用于指定结果集的限定条件。如果只需要处理数据表中特定数据时，可以通过该选项设定结果集限定条件
persister	用于指定持久化实现类
batch-size	用于指定每次批量操作的数量
optimistic-lock	用于指定采用乐观锁策略。所谓乐观锁是指乐观地认为每个事务在操纵数据时，很少有其他事务同时访问该数据资源。因而不在数据库层锁定，而是通过应用程序逻辑来实现锁定控制。属性值为 none（无乐观锁）、version（通过版本机制实现乐观锁）、dirty（通过检查发生变动过的属性实现乐观锁）和 all（通过检查所有属性实现乐观锁）

续表

属　性	说　明
lazy	用与指定是否采用延迟加载模式。延迟加载是指在用到时，才会获取被关联的对象，在非延迟加载的情况下，会自动获取被关联的对象。采用延迟加载在某种情况下，会减少无谓的性能开销
entity-name	指定实体类名。用于动态模型（Dynamic Model）支持
check	指定一个 SQL 语句，用于为自动生成的 schema 添加多行（multi-row）约束检查
rowid	Hibernate 可以使用数据库支持的所谓的 ROWIDs。例如，Oracle 数据库，如果你设置这个可选的 rowid， Hibernate 可以使用额外的字段 rowid 实现快速更新。ROWID 是这个功能实现的重点， 它代表了一个存储元组（tuple）的物理位置
subselect	它将一个不可变（immutable）并且只读的实体映射到一个数据库的子查询中。如果想用视图代替一张数据表时，该属性很有用，但最好不要这样做
abstract	用于在<union-subclass>的继承结构 （hierarchies）中标识抽象超类

示例

创建数据表 tb_user 与对应的实体类 UserForm 的映射关系，并指定采用延迟加载模式。

```
<class name="UserForm" table="tb_user" lazy="true">
</class>
```

示例

创建数据表 tb_user 与对应的实体类 UserForm 的映射关系，并采用乐观锁策略。

```
<class name="UserForm" table="tb_user" optimistic-lock="version">
</class>
```

示例

创建数据表 tb_user 与对应的实体类 UserForm 的映射关系，并设定每次批量操作的数量为 10。

```
<class name="UserForm" table="tb_user" batch-size="10" >
</class>
```

19.3 <id>节点——标识字段

<id>节点为<class>节点的子节点。在 Hibernate 中，要求必须在被映射的类中定一个与数据库中对应表中的主键字段相对应的属性，这个属性在映射文件中就是通过<id>节点进行描述的。

语法

```
<id
        name="propertyName"
        type="typename"
```


PDF电子书基地 http://dayo1982.400gb.com

```
        column="column_name"
        unsaved-value="null|any|none|undefined|id_value"
        access="field|property|ClassName">
    </id>
```

属性

<id>节点属性说明如下表所示。

<id>节点属性说明

属　性	说　明
name	用于指定映射类中对应主键的属性名
type	用于指定 Hibernate 映射类型。Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁，三者之间的对应关系如下表所示
column	用于指定主键字段名，默认值为当前类名
unsaved-value	用于标识对象是否已经保存的判定值。该属性在 Hibernate 3 中几乎不再需要
access	设置类属性访问方式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类）。多数情况下使用 property

Java 类型、Hibernate 映射类型和 SQL 类型三者之间的对应关系

Java 属性类型	Hibernate 映射类型	SQL 类型
String	string	varchar()
String	text	text
int	int	int
char	character	char()
boolean	boolean	bit
byte[]	binary	blob
java.sql.Date	date	date
java.sql.Timestamp	timestamp	Timestamp
float	float	float

示例

定义<id>节点，用于指定映射类中对应的主键属性，该属性的类型为 int，并且为该主键设置自动增值。

```
<id name="id" column="id" type="int">
        <generator class="increment"/> <!--设置自动增值-->
</id>
```

示例

定义<id>节点，用于指定映射类中对应的主键属性，该属性的类型为 int，

并且设置属性访问方式为 property。

```
<id name="id" column="id" type="int" access="property">
</id>
```

19.4 <generator>节点——主键生成策略

<generator>节点为<id>节点的子节点。<generator >节点用于为这这个持久化类的实例生成一个惟一标识。如果这个生成器实例需要某些配置值或者初始化参数，可以使用<param>元素来进行设置。

语法

```
<generator class="org.hibernate.id.TableHiLoGenerator">
        <param name="table">uid_table</param>
        <param name="column">next_hi_value_column</param>
</generator>
```

属性

<generator>节点只有一个 class 属性，该属性的属性值为 Hibernate 提供的内置标识符生成器，如下表所示。

Hibernate 提供的内置标识符生成器

标识符生成器	说　明
increment	适用于代理主键。由 Hibernate 以自增的方式生成，增量为 1
identity	适用于代理主键。由底层数据库生成，前提是底层数据库支持自增字段类型
sequence	适用于代理主键。Hibernate 根据底层数据库的序列生成，前提条件是底层数据库支持序列
hilo	适用于代理主键。Hibernate 根据 high/low 算法生成，Hibernate 把特定表的字段作为“heigh”值，在默认的情况下选用 hibernate_unique_key 表的 next_hi 字段
native	适用于代理主键。根据底层数据库对自动生成标识符的支持能力，来选择 identity、sequence、hilo
uuid.hex	适用于代理主键。Hibernate 采用 128 位的 UUID（Universal Unique Identification）算法生成，UUID 算法能够在网络环境生成唯一的字符串标识符，不推荐使用，因为字符串型要比整型占用更多的数据库空间
assigned	适用于自然主键。由 Java 应用程序负责生成，此时不能把 setId()方法声明为 private 类型，不推荐使用

示例

设置主键自动增值方式为由 Hibernate 以自增的方式生成。

```
<id name="id" column="id" type="int">
        <generator class="increment"/> <!--设置自动增值-->
</id>
```


PDF电子书基地 http://dayo1982.400gb.com

注意

在应用该方式设置自动增值时，数据库中对应的字段不能是自动编号，否则将产生错误。

示例

设置主键自动增值方式为由底层数据库生成，前提是底层数据库支持自增字段类型，例如，使用 SQL Server 数据库时，可以设置其标识。

```
<id name="id" column="id" type="int">
        <generator class="identity "/> <!--设置自动增值-->
</id>
```

19.5 <key>节点——映射外键

<key>节点用于在被关联数据表中定义了一个外键引用原表的主键。

语法

```
<key
        column="columnname"
        on-delete="noaction|cascade"
        property-ref="propertyName"
        not-null="true|false"
        update="true|false"
        unique="true|false"
>
</key>
```

属性

<key>节点属性说明如下表所示。

<key>节点属性说明

属　性	说　明
column	用于指定外键字段的名称
on-delete	用于设置外键关联是否打开级联删除。其属性值为 noaction（不打开级联删除）和 cascade（打开级联删除）
property-ref	用于指定外键引用的字段不是原表的主键
not-null	用于指定外键引用的字段是否允许为空。为 true 时，表示不允许为空，为 false 时，表示允许为空
update	用于指定外键引用的字段是否允许更新
unique	用于指定外键是否应具有唯一性

示例

为报损表与部门表设置一对多关联，指定其外键字段为 branchid。

```
<set name="damage" lazy="false" cascade="all" inverse="true">
    <key column="branchid"/>
    <one-to-many class="com.actionForm.DamageForm"/>
</set>
```

示例

为报损表与部门表设置一对多关联，指定其外键字段为 branchid，并且该外键不允许为空。

```
<set name="damage" lazy="false" cascade="all" inverse="true">
    <key column="branchid" not-null="true"/>
    <one-to-many class="com.actionForm.DamageForm"/>
</set>
```

19.6 <property >节点——映射实体属性

<property>节点是<class>节点的子节点，用于设定持久化类的属性的映射关系。

语法

```
<property
        name="propertyName"
        column="column_name"
        type="typename"
        update="true|false"
        insert="true|false"
        formula="arbitrary SQL expression"
        access="field|property|ClassName"
        lazy="true|false"
        unique="true|false"
        not-null="true|false"
        optimistic-lock="true|false"
        generated="never|insert|always"
/>
```

属性

<property>节点属性说明如下表所示。

< property >节点属性说明

属　性	说　明
name	用于指定属性名称，以小写字母开头
column	用于指定与属性名对应的字段名
type	用于指定 Hibernate 映射类型。Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁，三者之间的对应关系如下表所示
update	用于指定 Update 操作时是否包含本字段的数据。默认值为 true，即包含本字段
insert	用于指定 Insert 操作时是否包含本字段的数据。默认值为 true，即包含本字段

PDF电子书基地 http://dayo1982.400gb.com

续表

属　性	说　明
formula	用于为映射类配置表达式属性。这个映射的属性没有对应的数据库字段，而是由数据表中的某些字段计算而来
access	设置属性值的读取方式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类）。多数情况下使用 property
lazy	设置这个字段是否采用延迟加载模式。其中，值为 true 时，表示该属性值不会被立即加载，只有在真正读取该属性值时，其值才会真正从数据库读取。该属性在处理大型字段时非常有用
unique	用于指定该字段是否应具有唯一性
not-null	用于指定该字段是否允许为空。为 true 时，表示不允许为空，为 false 时，表示允许为空
optimistic-lock	用于指定在进行 Update 操作时是否需要获得乐观锁。默认值为 true
generated	用于指定该属性值是否由数据库生成。其属性值包括 never （表示此属性值不是从数据库中生成）、insert（表示此属性值在执行 insert 操作时生成，但不会在随后执行的 update 操作时重新生成）和 always（表示此属性值在执行 insert 和 update 操作时都会被生成）

Java 类型、Hibernate 映射类型和 SQL 类型三者之间的对应关系

Java 属性类型	Hibernate 映射类型	SQL 类型
String	string	varchar()
String	text	text
int	int	int
char	character	char()
boolean	boolean	bit
byte[]	binary	blob
java.sql.Date	date	date
java.sql.Timestamp	timestamp	Timestamp
float	float	float

示例

定义一个名称属性 name，对应的字段为 name，映射类型为 string，在执行 Insert 和 Update 操作时都包含该字段，并且不允许为空。

```
<property name="name" column="name" type="string" update="true" insert="true" not-null="true"/>
```

示例

定义一个金额属性 sellSum，该属性由数据库表中的单价 price 和数量 number 相乘计算得出。

```
<property name="sellSum" type="double" foumula="price*number"/>
```

示例

定义一个名称属性 memo，对应的字段为 memo，映射类型为 string，该字段采用延迟加载。

```
<property name="memo" column="memo" type="string" lazy="true"/>
```

19.7 <one-to-one>节点——一对一关联关系

<one-to-one>节点为<class>节点的子节点，用于映射持久化对象之间的一对一关联关系。

语法

```
<one-to-one
        name="propertyName"
        class="ClassName"
        cascade="cascade_style"
        constrained="true|false"
        fetch="join|select"
        property-ref="propertyNameFromAssociatedClass"
        access="field|property|ClassName"
        formula="any SQL expression"
        lazy="proxy|no-proxy|false"
        entity-name="EntityName"
/>
```

属性

<one-to-one>节点属性说明如下表所示。

<one-to-one>节点属性说明

属　性	说　明
name	用于指定属性名称
class	用于指定被关联类的名字
cascade	用于指定操作时的级联关系。该属性的属性值包括 none（在保存、修改或删除对象时，不对与之关联的对象进行任何操作）、save-update（在保存或修改对象时，级联保存或更新所有与之关联的对象）、delete（当删除本对象时，级联删除所有与之关联的对象）、all（包括 save-update 和 delete 的行为）
constrained	用于指定该映射文件所对应的映射表，和被关联的类所对应的数据表之间，通过一个外键引用对主键进行约束。如果值为 true，则表示该映射文件所映射表的主键同时作为外键，参照关联类对应表的主键
fetch	用于指定数据抓取策略。其属性值包括 join（外连接抓取）和 select（序列选择抓取）
property-ref	用于指定关联类的属性名。该属性将会和本类的主键相对应。如果没有指定，会使用对方关联类的主键

PDF电子书基地 http://dayo1982.400gb.com

续表

属　性	说　明
access	用于设置属性值的读取方式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类）。多数情况下使用 property
formula	绝大多数一对一的关联都指向其实体的主键。但是在某些情况下，可能会指向其他的一个或多个字段，或者是一个表达式，这时可以用一个 SQL 公式来表示
lazy	用于指定是否采用延迟加载。值为 true 时，表示采用延迟加载，值为 false 时，表示不采用延迟加载
entity-name	用于指定被关联类的实体名

示例

为用户信息表（对应的持久化类为 UserInfo）与权限信息表（对应的持久化类为 PurviewPK）映射一对一关联关系。

在 PurviewPK 类的映射文件中映射 UserInfo 类的实例。

```
<one-to-one name="userInfo" class="com.model.UserInfo" constrained="true" lazy="false"/>
```

在 UserInfo 类的映射文件中映射 PurviewPK 类的实例。

```
<one-to-one name="purviewPK" class="com.model.PurviewPK" cascade="all" lazy="false"/>
```

19.8 <many-to-many>节点——多对多关联关系

<many-to-many>节点用于映射持久化对象之间的多对多关联关系。

语法

```
<many-to-many
        column=" column_name "
        class=" ClassName "
        fetch="join|select"/>
```

属性

<many-to-many>节点属性说明如下表所示。

<many-to-many>节点属性说明

属　性	说　明
column	用于指定中间映射表中映射到关联目标表的关联字段
class	用于指定关联目标类的名称
fetch	用于指定数据抓取策略。其属性值包括 join（外连接抓取）和 select（序列选择抓取）

示例

为学生表(对应的持久化类为 Student)和课程表(对应的持久化类为 Lesson)建立多对多关联关系。

```
<set name="lesson" table="tb_lesson_student" lazy="false" sascade="save-update">
    <key column="student_id"/>
    <many-to-many class="com.model.Lesson" column="lesson_id"/>
</set>
```

说明

在关系型数据库中，无法直接建立学生表和课程表之间的多对多的关系，所以需要创建一个连接表，这里为 tb_lesson_student。

19.9 <set>节点——映射集合

<set>节点为<class>节点的子节点，用于映射持久化对象之间的一对多关联关系。

语法

```
<set
        name="propertyName"
        table="table_name"
        schema="schema_name"
        lazy="false|true"
        inverse="true|false"
        cascade="none|all|save-update|delete|all-delete-orphan"
        sort="unsorted|natural\comparatorClass"
        order-by="column_name asc|desc"
        where="arbitrary sql where condition"
        fetch="join|select"
        batch-size="N"
        access="property|field|ClassName"
>
</set>
```

属性

<set>节点属性说明如下表所示。

<set>节点属性说明

属　性	说　明
name	用于指定映射类的属性名称
table	用于指定关联的目标数据表
schema	用于指定数据库的 schema
lazy	用于指定是否采用延迟加载。值为 true 时，表示采用延迟加载；值为 false 时，表示不采用延迟加载
inverse	用于指定标识双向关联中的被方。属性值为 false 的一方（即主控方）负责维护关联关系

PDF电子书基地 http://dayo1982.400gb.com

续表

属　性	说　明
cascade	用于指定操作时的级联关系。该属性的属性值包括 none（在保存、修改或删除对象时，不对与之关联的对象进行任何操作）、save-update（在保存或修改对象时，级联保存或更新所有与之关联的对象）、delete（当删除本对象时，级联删除所有与之关联的对象）、all（包括 save-update 和 delete 的行为）
sort	用于设置排序的类型。该属性值包括 unsorted（不排序）、natural（自然排序）和 comparatorClass（指定某个实现了 java.util.Comparator 接口的类叙作为排序算法）
order-by	用于指定排序字段及排序方式，对应于 SQL 语句中的 order by 子句
where	用于指定添加筛选条件。这里设定的条件将被添加到所有的 SQL 查询语句中
fetch	用于指定数据抓取策略。其属性值包括 join（外连接抓取）和 select（序列选择抓取）
batch-size	用于指定采用延迟加载时，一次读取数据的数量
access	设置属性值的读取方式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类）。多数情况下使用 property

示例

为部门表（对应的持久化类为 BranchForm）与报损表（对应的持久化类为 DamageForm）设置一对多关联，不采用延迟加载，并且指定部门表为关联的主控方。

部门表对应的映射文件的配置代码如下：

```
<set name="damage" lazy="false" cascade="all" inverse="true">
    <key column="branchid"/>
    <one-to-many class="com.actionForm.DamageForm"/>
</set>
```

报损表对应的映射文件的配置代码如下：

```
<many-to-one name="branch" column="branchid" class="com.actionForm.BranchForm"
cascade="none" fetch="join" lazy="false"/>
```

19.10 <many-to-one>节点——多对一关联关系

<many-to-one>节点用于映射持久化对象之间的多对一关联关系。

语法

```
<many-to-one
name="propertyName"
column="column_name"
class="ClassName"
cascade="cascade_style"
fetch="join|select"
update="true|false"
insert="true|false"
```

```
        property-ref="propertyNameFromAssociatedClass"
        access="field|property|ClassName"
        unique="true|false"
        not-null="true|false"
        optimistic-lock="true|false"
        lazy="proxy|no-proxy|false"
        not-found="ignore|exception"
        entity-name="EntityName"
        formula="arbitrary SQL expression"
/>
```

属性

<many-to-one>节点属性说明如下表所示。

<many-to-one>节点属性说明

属　性	说　明
name	用于指定映射类的属性名称
column	用于指定关联的字段
class	用于指定关联类的名字
cascade	用于指定操作时的级联关系。该属性的属性值包括 none（在保存、修改或删除对象时，不对与之关联的对象进行任何操作）、save-update（在保存或修改对象时，级联保存或更新所有与之关联的对象）、delete（当删除本对象时，级联删除所有与之关联的对象）、all（包括 save-update 和 delete 的行为）
fetch	用于指定数据抓取策略。其属性值包括 join（外连接抓取）和 select（序列选择抓取）
update	用于指定执行 update 操作时是否包含此字段
insert	用于指定执行 insert 操作时是否包含此字段
property-ref	用于指定与主控类相关联的属性名，默认为关联类的主键属性名
access	设置属性值的读取方式。该属性的属性值包括 property（通过 setter 和 getter 方法对映射的属性进行存取操作）、field（直接访问持久化类的属性）和 ClassName（直接访问持久化类）。多数情况下使用 property
unique	用于指定使用 DDL 为外键字段生成一个唯一约束
not-null	用于指定使用 DDL 为外键字段生成一个非空约束
optimistic-lock	指定这个属性在做更新时是否需要获得乐观锁定
lazy	用于指定此关联对象是否采用延迟加载及延迟加载策略。其属性值包括 proxy（单向关联是经过代理）、no-proxy（在实例变量第一次被访问时采用延迟抓取，此时需要运行时字节码的增强）和 false（不采用延迟加载，即关联总是被预先抓取）
not-found	用于指定指定外键引用的数据不存在时的处理方式。属性值包括 ignore（将行数据不存在视为一个空（null）关联）和 exception（产生异常）
entity-name	用于指定被关联类的实体名
formula	用于定义计算出的外键值的 SQL 表达式

PDF电子书基地 http://dayo1982.400gb.com

示例

为员工表（对应的持久化类为 WorkerForm）与部门表（对应的持久化类为 BranchForm）设置多对一关联，不采用延迟加载，并且指定部门表为关联的主控方。

员工表对应的映射文件的配置代码如下：

```
<many-to-one name="branch" column="branchid" class="com.actionForm.BranchForm" cascade="none" fetch="join" lazy="false"/>
```

部门表对应的映射文件的配置代码如下：

```
<set name="worker" lazy="false" cascade="all" inverse="true">
    <key column="branchid"/>
    <one-to-many class="com.actionForm.WorkerForm"/>
</set>
```

19.11 <one-to-many>节点——一对多关联关系

<one-to-many>节点用于在映射一对多关联时，指定要关联的类。

语法

```
<one-to-many
        class="ClassName"
        entity-name="EntityName"
/>
```

属性

<one-to-many>节点属性说明如下表所示。

<one-to-many>节点属性说明

属　性	说　明
class	用于指定关联类的名字
entity-name	用于指定被关联类的实体名

示例

在映射员工表（对应的持久化类为 WorkForm）与部门表（对应的持久化类为 BranchForm）的一对多关联时，通过<one-to-many>节点指定部门表要关联的类为员工表所对应的类。

部门表对应的映射文件的配置代码如下：

```
<set name="work" lazy="false" cascade="all" inverse="true">
    <key column="branchid"/>
    <one-to-many class="com.actionForm.WorkForm"/>
</set>
```

第 20 章　Spring 容器

20.1　ClassPathXmlApplicationContext 类——创建容器

ClassPathXmlApplicationContext 类位于 org.springframework.context.support 包中，它是 ApplicationContext 接口的实现类之一，它从当前类路径中检索配置文件并装载它来创建容器的实例。

● 构造方法——ClassPathXmlApplicationContext 类的构造方法

ClassPathXmlApplicationContext 类的常用构造方法如下。

方法声明 1

```
public ClassPathXmlApplicationContext(String configLocation)
                                        throws BeansException
```

入口参数：configLocation 是 Spring 配置文件的路径和名称。

抛出异常：BeansException。

示例

使用指定的配置文件名称创建一个 ClassPathXmlApplicationContext 类的容器对象。

```
ClassPathXmlApplicationContext context;
context = new ClassPathXmlApplicationContext("appContext.xml");
```

方法声明 2

```
public ClassPathXmlApplicationContext(String[] configLocations)
                                        throws BeansException
```

入口参数：configLocations 是一个字符串数组，它包含了多个 Spring 配置文件的名称和位置。

抛出异常：BeansException。

示例

使用多个指定的配置文件的名称和路径创建一个字符串数组，然后使用该数组创建 ClassPathXmlApplicationContext 类的容器对象。

```
public static void main(String[] args) {
    String[] locations = {"appContext.xml", "daoConfig.xml", "viewConfig"};
    ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext(locations);
}
```


PDF电子书基地 http://dayo1982.400gb.com



20.2 FileSystemXmlApplicationContext 类——指定配置文件的位置

FileSystemXmlApplicationContext 类位于 org.springframework.context.support 包中，它同样是 ApplicationContext 接口的实现类，和 ClassPathXmlApplicationContext 类的区别在于读取 Spring 配置文件的方式。它不再从类路径中获取配置文件，而是通过参数指定配置文件的位置，可以获取类路径之外的资源。

构造方法——FileSystemXmlApplicationContext 类的构造方法

方法声明 1

```
public FileSystemXmlApplicationContext(String configLocation)
                                        throws BeansException
```

入口参数：configLocation 是 Spring 配置文件的位置和名称的字符串表现形式。

抛出异常：BeansException。

示例

使用指定的配置文件路径和名称创建 FileSystemXmlApplicationContext 类的容器对象。

```
public static void main(String[] args) {
    FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext("d:\\appContext.xml");
}
```

方法声明 2

```
public FileSystemXmlApplicationContext(String[] configLocations)
                                        throws BeansException
```

入口参数：configLocations 是一个字符串数组，它包含了多个 Spring 配置文件的名称和位置。

抛出异常：BeansException。

示例

使用多个指定的配置文件的名称和路径创建一个字符串数组，然后使用该数组创建 FileSystemXmlApplicationContext 类的容器对象。

```
public static void main(String[] args) {
    String[] locations = {"appContext.xml", "daoConfig.xml", "viewConfig"};
    FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext(locations);
}
```

20.3 XmlBeanFactory 类——解析 Spring 的配置

XmlBeanFactory 类位于 org.springframework.beans.factory.xml 包。它是 Spring 容器的实现类之一，它从 XML 文件中解析 Spring 的配置信息。

构造方法——XmlBeanFactory 类的构造方法

方法声明

```
public XmlBeanFactory(Resource resource)
                    throws BeansException
```

入口参数：resource 是 Resource 接口的实现，它包含配置文件的名称和位置等信息。

抛出异常： BeansException。

示例

使用 Resource 资源对象作为参数创建容器类对象。

```
public static void main(String[] args) {
        Resource res=new ClassPathResource("appContext.xml");
        XmlBeanFactory context = new XmlBeanFactory(res);
}
```

20.4 WebApplicationContext 接口——初始化 Spring 容器

WebApplicationContext 接口位于 org.springframework.web.context 包，它是 ApplicationContext 接口的扩展，主要应用于 Web 项目中。它从 Web 根目录的相对路径中装载配置文件来初始化 Spring 容器。

典型应用

为了说明 WebApplicationContext 的使用，这里通过实现一个简单的登录功能来介绍 Spring 的 MVC 框架，在这个示例中只使用到了 Spring MVC 部分的初级功能。

（1）要使用 Spring Web MVC，首先要在 Web 应用中的 web.xml 配置文件中定义 DispatcherServlet。web.xml 配置文件内容如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"能
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>WEB-INF/beans-config.xml</param-value>
        </init-param>
```


PDF电子书基地 http://dayo1982.400gb.com

```
            <load-on-startup>1</load-on-startup>
        </servlet>
            <servlet-mapping>
            <servlet-name>dispatcherServlet</servlet-name>
            <url-pattern>*.do</url-pattern>
        </servlet-mapping>
</web-app>
```

（2）建立登录页面 index.jsp 文件。在页面中建立两个用于输入用户名和密码的文本框和一个登录按钮。index.jsp 文件内容如下。

```
<%@page contentType="text/html" import="java.util.*,java.net.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Welcome</title>
    </head>
    <body>
        欢迎来到本页面，这里包含一个登录信息。请填写用户名和密码进行登录。
        <form name="form1" action="login.do" method="GET">
            <p>
                用户名：
                <input type="text" name="userName">
            </p>
            密  码：
            <input type="password" name="password"> 
            <input type="submit" name="login" value="登录">
        </form>
    </body>
</html>
```

（3）建立处理登录业务逻辑的控制器 LoginAction 类。它实现 Spring 的 org.springframe work.web.servlet.mvc.Controller 接口，并实现接口包含的处理请求的 handleRequest()方法。当控制器 LoginAction 收到分发器分配的请求时，会执行控制器的 handleRequest()方法完成登录的业务逻辑，判断用于登录的用户名和密码是否正确，如果登录信息正确，就转到登录成功页面，否则转到失败页面。控制器 LoginAction 类程序代码如下。

```
package action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class LoginAction implements Controller {
    private String fail_view;
    private String success_view;
    private String userName;
    private String password;
```

```
    public String getFail_view() {
        return fail_view;
    }
    public String getSuccess_view() {
        return success_view;
    }
    public void setFail_view(String string) {
        fail_view = string;
    }
    public void setSuccess_view(String string) {
        success_view = string;
    }
    public ModelAndView handleRequest(HttpServletRequest request,
            HttpServletResponse response) throws Exception {
        userName = request.getParameter("userName");
        password = request.getParameter("password");
        if (userName.equals("lzw") && password.equals("123"))
            return new ModelAndView(success_view);
        else
            return new ModelAndView(fail_view);
    }
}
```

（4）在 WEB-INF 中，建立 web.xml 中指定的 Spring 使用的 Bean 配置文件 beans-config。

xml 在配置文件中定义一个 Bean 为控制器 LoginAction 类和 viewResolver 用于视图解析的 Bean。关于视图解析将在后续章节讲解。beans-config.xml 文件内容如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:lang="http://www.springframework.org/schema/lang"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang-2.5.xsd">
    <bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/</value>
        </property>
        <property name="suffix">
            <value>.html</value>
        </property>
</bean>
<bean name="/login.do" class="action.LoginAction">
    <property name="fail_view">
        <value>failure</value>
    </property>
```


PDF电子书基地 http://dayo1982.400gb.com

```
        <property name="success_view">
            <value>success</value>
        </property>
    </bean>
</beans>
```

（5）在 WEB-INF 文件夹下的 jsp 子文件夹中建立用于显示登录成功的页面文件 success.jsp 文件和显示登录失败页面的 failure. jsp 文件。

success.jsp 文件内容如下。

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>登录成功页面</title>
    </head>
    <body>
        恭喜您，登录成功。
    </body>
</html>
```

failure.jsp 文件内容如下。

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>登录失败页面</title>
    </head>
    <body>
        对不起，您输入的用户名或密码不正确。
    </body>
</html>
```

登录页面的运行效果如图 20.1 所示。

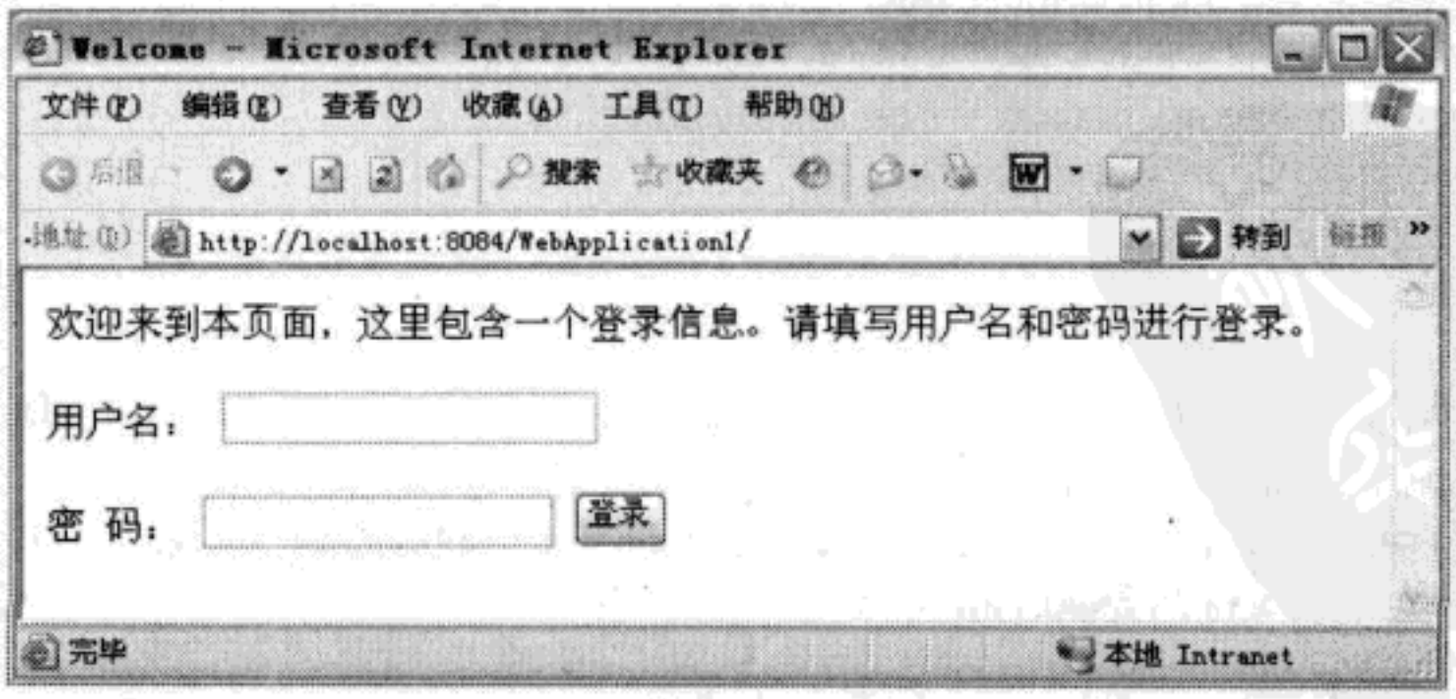


图 20.1 登录页面

在页面中输入用户名“lzw”，输入密码“123”，单击登录按钮向控制器“/login.do”发送请求。控制器会判断登录的用户名和密码，然后返回登录成功页面 success.jsp，如图 20.2 所示。

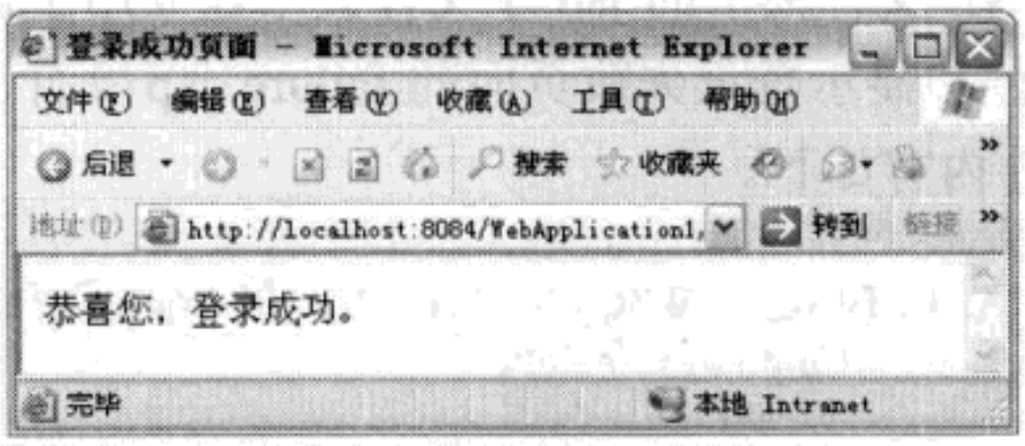


图 20.2 登录成功页面

20.5 Spring 容器类的方法——容器类

Spring 的容器类都是 Application 或者 BeanFactory 接口的实现，所以它们拥有共同的接口方法。这些常用的方法可以完成容器的业务处理。

20.5.1 containsBean 方法——判断是否包含指定名称 JavaBean

该方法用于查询当前容器是否包含指定名称的 JavaBean。

方法声明

```
public boolean containsBean(String name)
```

入口参数：name 是要查询的 JavaBean 的名称。

返回值：如果容器中包含指定名称的 JavaBean 返回 true，否则返回 false。

示例

判断 XmlBeanFactory 容器中是否包含名称为 dao 的 JavaBean。

```
public static void main(String[] args) {
    Resource res = new ClassPathResource("appContext.xml");
    XmlBeanFactory context = new XmlBeanFactory(res);
    context.containsBean("dao");
}
```

示例

判断 ApplicationContext 上下文容器中是否包含名称为 dao 的 JavaBean。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    context.containsBean("dao");
}
```


PDF电子书基地 http://dayo1982.400gb.com



20.5.2 getBean 方法——获取容器中的 JavaBean

该方法用于获取容器中的 JavaBean，这些 JavaBean 是由配置文件定义的。

方法声明 1

```
public Object getBean(String name)
                    throws BeansException
```

入口参数：name 是要获取的 JavaBean 的名字。

返回值：容器中指定名称的 JavaBean 对象。

抛出异常：BeansException。

示例

初始化 ApplicationContext 容器对象，并从容器中获取名称为 dao 的 JavaBean 对象。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");          // 初始化容器对象
    Object bean = context.getBean("dao");     // // 从容器中获取 dao 对象
}
```

方法声明 2

```
public Object getBean(String name, Class requiredType)
                    throws BeansException
```

入口参数：

name 是要获取的 JavaBean 的名字。

requiredType 是要获取的 JavaBean 的类型。

返回值：容器中指定名称的 JavaBean 对象。

抛出异常：BeansException。

示例

初始化 ApplicationContext 容器对象，并从容器中获取名称为 dao 的 JavaBean 对象，该 JavaBean 必须是 Date 类的示例对象。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    Object bean = context.getBean("time",Date.class);
    System.out.println("bean="+bean);
}
```

20.5.3 getBeanDefinitionCount 方法——获取 JavaBean 数量

该方法用于获取当前 Spring 容器中所定义的 JavaBean 的数量。

方法声明

```
public int getBeanDefinitionCount()
```

返回值：返回容器中的 JavaBean 数量。

示例

输出 Spring 容器中的 JavaBean 数量。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    int beanCount = context.getBeanDefinitionCount();
    System.out.println("容器中定义了"+beanCount+"个 JavaBean");
}
```

20.5.4 getBeanDefinitionNames 方法——获取 JavaBean 名称

该方法用于获取 Spring 容器中定义的所有 JavaBean 的名称。

方法声明

```
public String[] getBeanDefinitionNames()
```

返回值：Spring 容器中的所有 JavaBean 名称，返回类型是一个字符串数组。

示例

输出 Spring 容器中的所有 JavaBean 的名称。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    String[] beanNames = context.getBeanDefinitionNames();
    for (String name : beanNames) {
        System.out.println("JavaBean 名称：" + name);
    }
}
```

20.5.5 getBeanNamesForType 方法——通过类型获取 Bean 名称

该方法用于获取 Spring 容器中指定类型的所有 JavaBean 的名称。

方法声明

```
public String[] getBeanNamesForType(Class type)
```

入口参数：type 是指定的类的 Class 实例。

返回值：指定类型的所有 JavaBean 名称。

示例

获取 Spring 容器中所有 Date 类的 JavaBean 的名称。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    String[] beanNames = context.getBeanNamesForType(Date.class);
    for (String name : beanNames) {
        System.out.println("JavaBean 名称：" + name);
```


PDF电子书基地 http://dayo1982.400gb.com

```
        }
    }
```

20.5.6 getBeansOfType 方法——获取指定类型的 JavaBean

该方法用于获取 Spring 容器中指定类型的所有 JavaBean 对象。

方法声明

```
public Map getBeansOfType(Class type)
                    throws BeansException
```

入口参数：type 是指定的类的 Class 实例。

返回值：容纳指定类型 JavaBean 对象的 Map 集合。

抛出异常：BeansException。

示例

首先获取容器中 Date 类的所有 JavaBean 名称，然后获取所有 Date 类的 JavaBean 的 Map 集合，在使用 JavaBean 名称分别获取实例对象，然后输出结果。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    String[] beanNames =
            context.getBeanNamesForType(Date.class);
    // 获取指定类型的所有 JavaBean 对象
    Map beans = context.getBeansOfType(Date.class);
    for (String name : beanNames) {
        Date bean = (Date) beans.get(name);
        System.out.println("名称为" + name + "的 JavaBean 输出结果为： "
                + bean);
    }
}
```

该示例的配置文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
    <bean id="time" class="java.util.Date"></bean>
    <bean id="time2" class="java.util.Date"></bean>
    <bean id="time3" class="java.util.Date"></bean>
</beans>
```

20.5.7 getType 方法——获取 JavaBean 的类型

该方法用于获取容器中指定名称的 JavaBean 的类型。

方法声明

```
public Class getType(String name)
                    throws NoSuchBeanDefinitionException
```

入口参数：name 是容器中的 JavaBean 的名称。

返回值：指定名称的 JavaBean 的类型。

抛出异常：NoSuchBeanDefinitionException。

示例

从 Spring 容器中获取名称为 time 的 JavaBean 的类型，并输出到控制台中。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    Class type = context.getType("time");
    System.out.println("名称为 time 的 JavaBean 的类型是：" + type.getName());
}
```

20.5.8 isSingleton 方法——判断是否定义 Singleton 模式

该方法用于判断指定名称的 JavaBean 在 Spring 容器中是否定义为 Singleton 模式。

方法声明

```
public boolean isSingleton(String name)
                    throws NoSuchBeanDefinitionException
```

入口参数：name 是容器中 JavaBean 的名称。

返回值：如果指定名称的 JavaBean 被定义为 Singleton 模式返回 true，否则返回。

抛出异常：false NoSuchBeanDefinitionException。

示例

判断 Spring 容器中的名称为 time 的 JavaBean 是否为 Singleton 模式。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    boolean singleton = context.isSingleton("time");
    System.out.println("名称为 time 的 JavaBean 是否设置为 singleton 模式：" + singleton);
}
```

20.5.9 isTypeMatch 方法——判断 JavaBean 是否匹配指定的类型

该方法用于判断容器中定义的 JavaBean 是否匹配指定的类型。

方法声明

```
public boolean isTypeMatch(String name,
                           Class targetType)
                    throws NoSuchBeanDefinitionException
```

入口参数：

name 是容器中定义的 JavaBean 的名称。

targetType 是要匹配的目标类型。

PDF电子书基地 http://dayo1982.400gb.com

返回值：如果 JavaBean 匹配指定的类型，返回 true，否则返回 false。

抛出异常：NoSuchBeanDefinitionException。

示例

从 Spring 容器中判断名称为 time 的 JavaBean 的类型是否为 java.uitl.Date 类。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "appContext.xml");
    boolean typeMatch = context.isTypeMatch("time", Date.class);
    System.out.println("名称为 time 的 JavaBean 类型是否 Date：" + typeMatch);
}
```

20.5.10 getServletContext 方法——获取 ServletContext 上下文

该方法用于获取当前 Web 应用程序的 ServletContext 上下文对象。

方法声明

```
ServletContext getServletContext()
```

返回值：当前 Web 应用程序的 ServletContext 上下文对象。

示例

在 Spring 的控制器处理方法中获取当前项目的 ServletContext 上下文对象。

```
public class LoginAction extends AbstractController {
    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest arg0,
            HttpServletResponse arg1) throws Exception {
        WebApplicationContext context = getWebApplicationContext();
        context.getServletContext();            // 获取 ServletContext 对象
        return null;
    }
}
```

第 21 章　Spring 持久化

21.1　DriverManagerDataSource 类——驱动管理器数据源

该类位于 Spring 的 org.springframework.jdbc.datasource 类包。这是标准 JDBC 数据源的一个简单实现类，它用于开发简单的应用和程序测试，并且不支持连接池，每次连接数据库都是创建新的连接对象。

21.1.1　构造方法——DriverManagerDataSource 类的构造方法

DriverManagerDataSource 类的构造方法如下。

方法声明 1

默认的 DriverManagerDataSource 构造方法。

```
public DriverManagerDataSource()
```

示例

使用默认的构造方法创建 DriverManagerDataSource 类的对象，然后设置连接属性。

```
DriverManagerDataSource driverManagerDataSource =
        new DriverManagerDataSource();
driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");
String url ="jdbc:mysql://lzw:3306/testDatabase";
driverManagerDataSource.setUrl(url);
driverManagerDataSource.setUsername("root");
driverManagerDataSource.setPassword("111");
```

方法声明 2

```
public DriverManagerDataSource(String url)
                        throws CannotGetJdbcConnectionException
```

入口参数：url 是访问数据库的 URL 路径。

抛出异常：CannotGetJdbcConnectionException。

示例

使用指定的 URL 参数创建 DriverManagerDataSource 类的实例对象，然后设置其他参数。

```
DriverManagerDataSource driverManagerDataSource =
        new DriverManagerDataSource(url);
driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");
```


PDF电子书基地 http://dayo1982.400gb.com




```
driverManagerDataSource.setUsername("root");
driverManagerDataSource.setPassword("111");
```

方法声明 3

```
public DriverManagerDataSource(String url,
                               String username,
                               String password)
                        throws CannotGetJdbcConnectionException
```

入口参数：

url 是访问数据库的 URL 路径。

username 是访问数据库的用户名。

password 是访问数据库的密码。

抛出异常：CannotGetJdbcConnectionException。

示例

使用访问数据库的 URL、用户名、密码等参数创建 DriverManagerDataSource 类的实例对象，然后设置数据库驱动类属性。

```
String url = "jdbc:mysql://lzw:3306/testDatabase";
String username = "root";
String password = "111";
DriverManagerDataSource driverManagerDataSource =
        new DriverManagerDataSource(url, username, password);
driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");
```

方法声明 4

```
public DriverManagerDataSource(String driverClassName,
                               String url,
                               String username,
                               String password)
                        throws CannotGetJdbcConnectionException
```

入口参数：

driverClassName 是驱动类的名称。

url 是访问数据库的 URL 路径。

username 是访问数据库的用户名。

password 是访问数据库的密码。

抛出异常：CannotGetJdbcConnectionException。

示例

使用参数创建完整的 DriverManagerDataSource 类的实例对象，不用再调用方法设置任何属性。

```
String driver="com.mysql.jdbc.Driver";
String url = "jdbc:mysql://lzw:3306/testDatabase";
String username = "root";
String password = "111";
DriverManagerDataSource driverManagerDataSource =
        new DriverManagerDataSource(driver,url, username, password);
```

21.1.2 getConnection 方法——获取数据库连接对象

该方法用于获取数据库连接对象，该对象用于数据库操作。

方法声明 1

```
public Connection getConnection()
                                throws SQLException
```

返回值：Connection 类的数据库连接对象。

抛出异常：SQLException。

示例

创建 DriverManagerDataSource 类的实例对象，并调用 getConnection 方法获取数据库连接对象。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    try {
        DriverManagerDataSource driverManagerDataSource =
            new DriverManagerDataSource(driver, url, username, password);
        Connection connection = driverManagerDataSource.getConnection();
        if(connection!=null)
            System.out.println("数据库连接成功");
            // 其他数据库操作
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

方法声明 2

使用指定的用户名和密码进行数据库连接，并返回连接对象。

```
public Connection getConnection(String username,
                                String password)
                         throws SQLException
```

入口参数：

username 是连接数据库的用户名。

password 是连接数据库的密码。

PDF电子书基地 http://dayo1982.400gb.com

返回值：Connection 类的数据库连接对象。

抛出异常：SQLException。

示例

创建并初始化 DriverManagerDataSource 类的实例对象，然后调用该对象的 getConnection 方法获取数据库连接，并输出数据库连接成功的信息。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    try {
        DriverManagerDataSource driverManagerDataSource =
                new DriverManagerDataSource(url);
        driverManagerDataSource.setDriverClassName(driver);
        Connection connection = driverManagerDataSource.getConnection(
          username, password);      // 使用指定的用户名和密码连接数据库
        if (connection != null) {
            System.out.println("数据库连接成功");
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

21.1.3 getConnectionProperties 方法——获取数据库连接属性

该方法用于获取数据源的数据库连接属性。

方法声明

```
public Properties getConnectionProperties()
```

返回值：获取数据库连接属性集，返回类型为 Properties

示例

创建并初始化 DriverManagerDataSource 类的实例对象，然后调用该对象的 getConnectionProperties()方法获取数据库连接属性集，如果属性集不为空，则输出所有属性信息。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
            url, username, password);
    Properties props = dmd.getConnectionProperties();
    if (props != null) {
```

```
            props.list(System.out);
        }
    }
```

21.1.4 getDriverClassName 方法——获取数据库驱动类的名称

该方法用于获取数据源设置的数据库驱动类的名称。

方法声明

```
public String getDriverClassName()
```

返回值：当前数据源设置的数据库驱动类的名称。

示例

创建并初始化 DriverManagerDataSource 类的实例对象，然后调用该对象的 getDriverClassName ()方法获取并输出数据库驱动类的名称。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
            url, username, password);
    String driverClassName = dmd.getDriverClassName();
    System.out.println("数据库驱动类为："+driverClassName);
}
```

21.1.5 getPassword 方法——获取数据库的密码

该方法用于获取数据源设置的访问数据库的密码。

方法声明

```
public String getPassword()
```

返回值：访问数据库的密码。

示例

创建并初始化 DriverManagerDataSource 类的实例对象，然后调用该对象的 getPassword ()方法获取并输出访问数据库的密码。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
            url, username, password);
    String psw = dmd.getPassword();
    System.out.println("访问数据库的密码是："+psw);
}
```


PDF电子书基地 http://dayo1982.400gb.com

21.1.6 getUrl 方法——获取数据库的 URL 路径

该方法用于获取数据源设置的访问数据库的 URL 路径。

方法声明

```
public String getUrl()
```

返回值：访问数据库的 URL 路径，返回类型是 String。

示例

从 DriverManagerDataSource 类的实例对象中获取访问数据库的 URL 路径。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    String username = "root";
    String password = "111";
    DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
            url, username, password);
    String accurl = dmd.getUrl();
    System.out.println("访问数据库的 URL 是："+accurl);
}
```

21.1.7 getUsername 方法——获取数据库的用户名称

该方法用于获取数据源访问数据库的用户名称。

方法声明

```
public String getUsername()
```

返回值：访问数据库的用户名。

示例

从 DriverManagerDataSource 类的实例对象中获取访问数据库的用户名。

```
public static void main(String[] args) {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://lzw:3306/testDatabase";
    DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
            url, "root", "111");
    String name = dmd.getUsername();
    System.out.println("访问数据库的用户名是："+name);
}
```

21.1.8 setConnectionProperties 方法——设置数据库连接属性

该方法用于设置数据库连接的各种属性。

方法声明

```
public void setConnectionProperties(Properties connectionProperties)
```

入口参数：connectionProperties 是连接数据库的属性集合对象，该对象是 Properties 类的实例对象。

示例

创建一个属性集合，将数据库连接需要的属性添加到该属性集合中，然后将该属性集合添加到 DriverManagerDataSource 类的实例对象中。

```
public static void main(String[] args) {
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://lzw:3306/testDatabase";
        DriverManagerDataSource dmd = new DriverManagerDataSource(driver,
                url, "root", "111");
        Properties props = new Properties();
        props.setProperty("useUnicode", "true");
        props.setProperty("characterEncoding", "GB2312");
        // 设置连接属性集
        dmd.setConnectionProperties(props);
        // 获取连接属性集并输出
        if (props != null) {
            props.list(System.out);
        }
}
```

21.1.9 setDriverClassName 方法——设置数据源的驱动类

该方法用于设置数据源的驱动类。

方法声明

```
public void setDriverClassName(String driverClassName)
                                throws CannotGetJdbcConnectionException
```

入口参数：driverClassName 是数据库驱动类的名称。

抛出异常：CannotGetJdbcConnectionException。

示例

创建 DriverManagerDataSource 类的实例对象，然后调用 setDriverClassName 方法设置驱动类。

```
public static void main(String[] args) {
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://lzw:3306/testDatabase";
        String username = "root";
        String password = "111";
        DriverManagerDataSource dmd =
                new DriverManagerDataSource(url);
        dmd.setDriverClassName(driver);// 设置数据库的驱动类
// 省略获取数据库连接和其他操作
}
```


PDF电子书基地 http://dayo1982.400gb.com

21.1.10 setPassword 方法——设置数据库的密码

该方法用于设置数据源连接数据库的密码。

方法声明

```
public void setPassword(String password)
```

入口参数：password 是连接数据库的密码。

示例

创建数据源对象，然后调用 setPassword 方法设置连接数据库的密码。

```
DriverManagerDataSource dmd = new DriverManagerDataSource();
dmd.setPassword(password);// 设置访问数据库的密码
```

21.1.11 setUrl 方法——设置连接数据库的 URL 路径

该方法用于设置数据源连接数据库的 URL 路径。

方法声明

```
public void setUrl(String url)
```

入口参数：url 是访问数据库的 URL 路径。

示例

创建数据源对象，然后调用 setUrl 方法设置数据源访问数据库的 URL 路径。

```
DriverManagerDataSource dmd =    new DriverManagerDataSource();
dmd.setUrl("jdbc:mysql://lzw:3306/testDatabase");
```

21.1.12 setUsername 方法——设置访问数据库的用户名

该方法用于设置数据源访问数据库的用户名。

方法声明

```
public void setUsername(String username)
```

入口参数：username 是访问数据库的用户名。

示例

创建数据源对象，然后调用 setUsername 方法设置数据源访问数据库的用户名称。

```
DriverManagerDataSource dmd = new DriverManagerDataSource();
dmd.setUsername("root");
```

21.2 HibernateTemplate 类——Hibernate 模板

该类位于 Spring 的 org.springframework.orm.hibernate3 包中，是 Spring 的 Hibernate 模板类，它封装了数据源并提供了多个模板方法，使 Hibernate 应用更加方便。

21.2.1 构造方法——HibernateTemplate 类的构造方法

方法声明 1

默认的 HibernateTemplate 类构造方法。

```
public HibernateTemplate()
```

示例

使用默认的构造方法创建 HibernateTemplate 类的实例对象。

```
HibernateTemplate htm=null;
htm= new HibernateTemplate();
```

方法声明 2

使用 Session 工厂创建 HibernateTemplate 类的对象的构造方法。

```
public HibernateTemplate(SessionFactory sessionFactory)
```

入口参数：sessionFactory 是 Hibernate 的 Session 工厂类的实例对象。

示例

获取 SessionFactory 的实例对象，然后使用该对象创建 HibernateTemplate 类的实例对象。

```
SessionFactory sf = …;                          // 省略 SessionFactory 的初始化
HibernateTemplate htm=new HibernateTemplate(sf);
```

方法声明 3

使用 Session 工厂创建 HibernateTemplate 类的对象，并同时指定是否允许创建 Session 的构造方法

```
public HibernateTemplate(SessionFactory sessionFactory,
                         boolean allowCreate)
```

入口参数：

sessionFactory 是 Hibernate 的 Session 工厂类的实例对象。

allowCreate 指定如果 Session 没有别创建的话，是否允许创建 Session。

示例

获取 SessionFactory 的实例对象，然后使用该对象创建 HibernateTemplate 类的实例对象，并指定允许创建 Session。

```
SessionFactory sf = …;                          // 省略 SessionFactory 的初始化
HibernateTemplate htm=new HibernateTemplate(sf , true);
```

21.2.2 delete 方法——删除的实体对象

delete 方法从数据库中删除指定的实体数据。

PDF电子书基地 http://dayo1982.400gb.com

方法声明

```
public void delete(Object entity)
        throws DataAccessException
```

入口参数：entity 是要删除的实体对象。

抛出异常：DataAccessException。

示例

如果用户实体对象的年龄属性小于 20，那么就从数据库删除指定的用户实体数据。

```
public void delBoyUser(TbUser user) {
    HibernateTemplate htm = getHibernateTemplate();
    if (user.getAge() < 20) {
        htm.delete(user);
    }
}
```

21.2.3 deleteAll 方法——删除指定集合中实体对象

该方法用于从数据库中删除指定集合中包含的所有实体对象。

方法声明

```
public void deleteAll(Collection entities)
        throws DataAccessException
```

入口参数：entities 是包含实体对象的集合。

抛出异常：DataAccessException。

示例

定制一个删除方法，使该方法能够删除指定集合对象中的所有实体对象。

```
public void delBoyUser(Collection users) {
    HibernateTemplate htm = getHibernateTemplate();
    htm.deleteAll(users);
}
```

21.2.4 execute 方法——接口回调执行操作

execute 方法用于执行更多的底层的 Hibernate 功能，它使用 HibernateCallback 接口回调执行操作。

方法声明

```
public Object execute(HibernateCallback action)
        throws DataAccessException
```

入口参数：action 是 HibernateCallback 接口的实现。

返回值：返回 Hibernate 查询结果，或更新记录的数量，可以是任意类型。

示例

使用 execute 方法调用更加底层的 Hibernate 操作，实现根据 ID 获取用户实体对象。

```
public TbUser getUser(final Integer ID) {
    HibernateTemplate htm = getHibernateTemplate();
    TbUser user = (TbUser) htm.execute(new HibernateCallback() {
        public Object doInHibernate(Session session) throws
                HibernateException,
                SQLException {
            Object user = session.get(TbUser.class, ID);
            return user;
        }
    });
    return user;
}
```

21.2.5 executeFind 方法——接口回调查询数据

该法用于执行 Hibernate 查询操作，但是它使用 HibernateCallback 接口回调，操作底层的 Hibernate 功能来实现，其返回值是一个 List 集合。

方法声明

```
public List executeFind(HibernateCallback action)
                    throws DataAccessException
```

入口参数：action 是 HibernateCallback 接口的实现。

返回值：Hibernate 的查询结果集，类型为 List 集合。

示例

编写 getSmallUser 方法获取年龄不足 18 岁的所有用户。

```
public List getSmallUser(final Integer ID) {
    HibernateTemplate htm = getHibernateTemplate();
    final String sql = "FROM TbUser u where u.age<18";
    List list = htm.executeFind(new HibernateCallback() {
        public Object doInHibernate(Session session) throws
                HibernateException,
                SQLException {
            Query query = session.createQuery(sql);
            List list = query.list();
            return list;
        }
    });
    return list;
}
```


PDF电子书基地 http://dayo1982.400gb.com

21.2.6 find 方法——执行 Hibernate 查询

find 方法用于执行 Hibernate 查询，它有多种重载格式。

方法声明 1

使用指定的 HQL 语句进行 Hibernate 查询。

```
public List find(String queryString)
        throws DataAccessException
```

入口参数：queryString 是查询数据的 HQL 语句。

返回值：方法的查询结果集，类型为 List。

示例

定义用户查询方法，查询用户表中的所有用户实体。

```
public List findUsers() {
    HibernateTemplate htm = getHibernateTemplate();
    String hql = "FROM tb_user";
    List find = htm.find(hql);
    return find;
}
```

方法声明 2

使用带参数的 HQL 语句和参数值执行 Hibernate 查询，HQL 语句中只能包含 1 个参数。

```
public List find(String queryString,
                 Object value)
          throws DataAccessException
```

入口参数：

queryString 是带“？”参数的 HQL 查询语句。

value 是 HQL 语句的参数值。

抛出异常：DataAccessException。

示例

获取用户表中所有女性用户的集合。

```
public List find() {
    HibernateTemplate htm = getHibernateTemplate();
    String hql = "FROM TbUser u where u.sex=?";
    List find = htm.find(hql,"女");
    return find;
}
```

方法声明 3

使用带参数的 HQL 语句和参数值执行 Hibernate 查询，HQL 语句中可以包含多个参数。

```
public List find(String queryString,
                 Object[] values)
          throws DataAccessException
```

入口参数：

queryString 是带“？”参数的 HQL 查询语句。

values 以数组容纳 HQL 语句的所有参数值。

抛出异常：DataAccessException。

示例

获取用户表中所有性别为男，年龄 27 的用户的集合。

```
public List find() {
    HibernateTemplate htm = getHibernateTemplate();
    String hql = "FROM TbUser u where u.sex=? and u.age=?";
    Object[] args={"男",new Integer(27)};
    List find = htm.find(hql,args);
    return find;
}
```

21.2.7 get 方法——获取指定 ID 编号的实体对象

get 方法用于获取指定 ID 编号的实体对象。

方法声明

```
public Object get(Class entityClass,
                  Serializable id)
           throws DataAccessException
```

入口参数：

entityClass 是实体类的 Class 实例。

id 是实体类的 ID 编号。

抛出异常：DataAccessException。

示例

定义根据 ID 编号获取用户实体对象的方法。

```
public TbUser getUser(Integer id) {
    HibernateTemplate htm = getHibernateTemplate();
    TbUser user = (TbUser) htm.get(TbUser.class, id);
    return user;
}
```

21.2.8 loadAll 方法——装载指定类型的所有实体对象

该方法用于装载指定类型的所有实体对象。

方法声明

```
public List loadAll(Class entityClass)
          throws DataAccessException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：entityClass 是实体类的 Class 实例。

抛出异常：DataAccessException。

示例

定义根据 ID 编号获取用户实体对象的方法。

```
public List loadUsers() {
    HibernateTemplate htm = getHibernateTemplate();
    List list = htm.loadAll(TbUser.class);
    return list;
}
```

21.2.9 save 方法——保存实体对象

save 方法用于保存实体对象到数据库中。

方法声明

```
public Serializable save(Object entity)
                    throws DataAccessException
```

入口参数：entity 是要保存的实体对象。

抛出异常：DataAccessException。

示例

创建一个用户实体对象，并使用 save 方法保存到数据库中。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    TbUser user = new TbUser();
    user.setName("明日");
    user.setSex("男");
    user.setAge(20);
    dao.getHibernateTemplate().save(user);
}
```

21.2.10 saveOrUpdate 方法——保存或更新实体对象

该方法可以自动判断执行实体对象的保存操作还是更新操作。

方法声明

```
public void saveOrUpdate(Object entity)
                    throws DataAccessException
```

入口参数：entity 是要保存的实体对象。

抛出异常：DataAccessException。

示例

创建一个新的用户实体对象，然后调用 saveOrUpdate 方法将执行实体对象的保存操作。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    TbUser user = new TbUser();
    user.setName("保存的用户");
    user.setSex("男");
    user.setAge(20);
    HibernateTemplate htm = dao.getHibernateTemplate();
    htm.saveOrUpdate(user);
}
```

示例

从数据库获取一个用户实体对象，然后调用 saveOrUpdate 方法将执行实体对象的更新操作。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    HibernateTemplate htm = dao.getHibernateTemplate();
    TbUser user = (TbUser) htm.get(TbUser.class, new Integer(1));
    if(user==null)
        return;
    user.setName("更新的用户");
    htm.saveOrUpdate(user);
}
```

21.2.11 saveOrUpdateAll 方法——保存或更新实体对象集合

该方法用于保存或更新用户实体对象的集合，它能根据实体对象的状态判断集合中的每个对象应该执行保存还是更新操作。

方法声明

```
public void saveOrUpdateAll(Collection entities)
                    throws DataAccessException
```

入口参数：entities 是要保存或更新的实体对象的集合。

抛出异常：DataAccessException。

示例

创建多个用户，并添加到一个集合对象中，然后把该集合中的所有实体对象保存到数据库中。

PDF电子书基地 http://dayo1982.400gb.com

```
public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        HibernateTemplate htm = dao.getHibernateTemplate();
        TbUser user1=new TbUser("用户 1", "男", 27);
        TbUser user2=new TbUser("用户 2", "男", 27);
        TbUser user3=new TbUser("用户 3", "男", 27);
        List list = new ArrayList();
        list.add(user1);
        list.add(user2);
        list.add(user3);
        htm.saveOrUpdateAll(list);
    }
```

示例

装载用户表的所有用户实体对象到集合中，然后修改部分或全部实体对象，通过 saveOrUpdateAll 方法更新集合中的所有实体对象。

```
public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        HibernateTemplate htm = dao.getHibernateTemplate();
        List list = htm.loadAll(TbUser.class);
        // …省略修改集合中的实体对象的代码
        htm.saveOrUpdateAll(list);
    }
```

21.2.12 update 方法——更新实体对象

该方法用于更新指定的实体对象。

方法声明

```
public void update(Object entity)
                throws DataAccessException
```

入口参数：entity 是要更新的实体对象。

抛出异常：DataAccessException。

示例

从数据库获取用户实体对象，修改该用户的年龄，然后更新到数据库中。

```
public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        HibernateTemplate htm = dao.getHibernateTemplate();
        TbUser user = (TbUser) htm.get(TbUser.class, 1);
        user.setAge(user.getAge()+1);
        htm.update(user);
}
```

21.3 JdbcTemplate 类——JDBC 模板

该类位于 Spring 的 org.springframework.jdbc.core 包中，是 Spring 的 JDBC 模板类，它封装了数据源并提供了多个模板方法，使 JDBC 应用更加方便。

21.3.1 batchUpdate 方法——批量更新语句

该方法用于执行数据库的批量更新语句。

方法声明

```
public int[] batchUpdate(String[] sql)
                        throws DataAccessException
```

入口参数：sql 是多条 SQL 语句组成的字符串数组。

抛出异常：DataAccessException。

示例

定义批量处理的 SQL 语句数组，通过 batchUpdate 方法一次执行。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String[] sqls = {
        "delete from tb_user where age<27",
        "delete from tb_user where age>35"
    };
    int[] batchUpdate = jtm.batchUpdate(args);
}
```

21.3.2 query 方法——用于 JDBC 查询

query 方法用于 JDBC 查询，它有多种重载方法。

方法声明 1

使用 SQL 语句和 RowCallbackHandler 回调接口，处理查询结果。

```
public void query(String sql,
                  RowCallbackHandler rch)
           throws DataAccessException
```

入口参数：

sql 是执行 JDBC 查询的 SQL 语句。

rch 是 RowCallbackHandler 回调接口的实现。

抛出异常：DataAccessException。

示例

定义单用户查询方法，从数据库中查询指定 ID 编号的用户，将查询结果赋

PDF电子书基地 http://dayo1982.400gb.com

值给用户实体类。

```
public TbUser getUser(int id) {
    final TbUser user = null;
    String sql = "select * from tb_user where id=" + id;
    getJdbcTemplate().query(sql, new RowCallbackHandler() {
        public void processRow(ResultSet rs) throws SQLException {
            user.setAge(rs.getInt("age"));
            user.setId(rs.getInt("id"));
            user.setName(rs.getString("name"));
            user.setSex(rs.getString("sex"));
        }
    });
    return user;
}
```

示例

定义用户查询方法，从数据库中查询所有用户，将查询结果赋值给用户实体对象并将所有实体对象添加到 List 集合中。

```
public List getAllUsers() {
    final List list=new ArrayList();
    String sql = "select * from tb_user";
    getJdbcTemplate().query(sql, new RowCallbackHandler() {
        public void processRow(ResultSet rs) throws SQLException {
            TbUser user=new TbUser();
            user.setAge(rs.getInt("age"));
            user.setId(rs.getInt("id"));
            user.setName(rs.getString("name"));
            user.setSex(rs.getString("sex"));
            list.add(user);
        }
    });
    return list;
}
```

方法声明 2

使用 SQL 语句、参数和 RowCallbackHandler 回调接口，处理查询结果。

```
public void query(String sql,
                  Object[] args,
                  RowCallbackHandler rch)
    throws DataAccessException
```

入口参数：

sql 是执行 JDBC 查询的 SQL 语句。

args 是 SQL 语句中对应的参数值数组。

rch 是 RowCallbackHandler 回调接口的实现。

抛出异常：DataAccessException。

示例

定义单用户查询方法，从数据库中查询指定 ID 编号和指定年龄的用户，将查询结果赋值给用户实体类。

```
public TbUser getUser(int id,int age) {
    final TbUser user = null;
    String sql = "select * from tb_user where id=? and age=?";
    Integer[] args={id,age};
    getJdbcTemplate().query(sql, args, new RowCallbackHandler() {
        public void processRow(ResultSet rs) throws SQLException {
            user.setAge(rs.getInt("age"));
            user.setId(rs.getInt("id"));
            user.setName(rs.getString("name"));
            user.setSex(rs.getString("sex"));
        }
    });
    return user;
}
```

方法声明 3

使用 SQL 语句、参数和 RowMapper 回调接口，处理查询结果。

```
public void query(String sql,
                  RowMapper rowMapper)
           throws DataAccessException
```

入口参数：

sql 是执行 JDBC 查询的 SQL 语句。

rowMapper 是 RowMapper 回调接口的实现。

抛出异常：DataAccessException。

示例

定义用户查询方法，从数据库中查询所有用户实体对象。

```
public List getAllUsers() {
    String sql = "select * from tb_user";
    List list = getJdbcTemplate().query(sql, new RowMapper() {
        public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
            TbUser user = new TbUser();
            user.setAge(rs.getInt("age"));
            user.setId(rs.getInt("id"));
            user.setName(rs.getString("name"));
            user.setSex(rs.getString("sex"));
            return user;
        }
    });
    return list;
}
```


PDF电子书基地 http://dayo1982.400gb.com



方法声明 4

使用 SQL 语句、参数和 RowMapper 回调接口，处理查询结果。

```
public void query(String sql,
                  Object[] args,
                  RowMapper rowMapper)
           throws DataAccessException
```

入口参数：

sql 是执行 JDBC 查询的 SQL 语句。

args 是 SQL 语句中对应的参数值数组。

rowMapper 是 RowMapper 回调接口的实现。

抛出异常：DataAccessException。

示例

定义单用户查询方法，从数据库中查询指定年龄段的用户，返回这些用户实体对象的 List 集合。

```
public List getUsers(int minAge, int maxAge) {
    String sql = "select * from tb_user where age>? && age<?";
    Integer[] args = {minAge, maxAge};
    List list = getJdbcTemplate().query(sql, args, new RowMapper() {
        public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
            TbUser user = new TbUser();
            user.setAge(rs.getInt("age"));
            user.setId(rs.getInt("id"));
            user.setName(rs.getString("name"));
            user.setSex(rs.getString("sex"));
            return user;
        }
    });
    return list;
}
```

21.3.3 queryForInt 方法——int 类型单值查询

该方法用于执行单个返回值的 SQL 查询，返回结果是 int 类型。

方法声明 1

```
public int queryForInt(String sql)
                throws DataAccessException
```

入口参数：sql 是查询语句。

返回值：int 类型的单值查询结果。

抛出异常：DataAccessException。

示例

编写获取用户总数的方法。

```
public int getUserCount() {
    String sql="SELECT count(*) FROM tb_user";
    int count = getJdbcTemplate().queryForInt(sql);
    return count;
}
```

方法声明 2

支持 SQL 查询参数的方法。

```
public int queryForInt(String sql,
                       Object[] args)
                throws DataAccessException
```

入口参数：

sql 是执行的 SQL 查询语句。

args 是 SQL 语句中对应的参数值。

返回值：int 类型的单值查询结果。

抛出异常：DataAccessException。

示例

查询所有年龄为 27 岁、性别为男的用户数量。

```
public int getUserCount() {
    String sql="SELECT count(*) FROM tb_user WHERE sex=? and age=?";
    Object[] args={"男",27};
    int count = getJdbcTemplate().queryForInt(sql,args);
    return count;
}
```

21.3.4 queryForList 方法——返回结果集合的 SQL 查询

该方法用于执行 SQL 查询，查询结果保存在 List 集合中。

方法声明 1

该方法用于 SQL 的多值查询，查询结果包含在方法返回的 List 集合中，集合中的每个元素都是 Map 集合对象，以数据表的字段名作为 Map 集合的 Key 名称。

```
public List queryForList(String sql)
                  throws DataAccessException
```

入口参数：sql 是查询数据库的 SQL 语句。

返回值：包含查询结果的 List 集合对象。

抛出异常：DataAccessException。

示例

获取并输出用户表所有用户信息。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
```


PDF电子书基地 http://dayo1982.400gb.com

```
                "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        JdbcTemplate jtm = dao.getJdbcTemplate();
        String sql="select * from tb_user";
            List list = jtm.queryForList(sql);
            for (Object object : list) {
                Map map=(Map) object;
                System.out.print("\tmap="+map.get("id"));
                System.out.print("\tmap="+map.get("name"));
                System.out.print("\tmap="+map.get("sex"));
                System.out.println("\tmap="+map.get("age"));
            }
    }
```

方法声明 2

执行单值查询的方法，多有查询结果以 List 集合返回。

```
public List queryForList(String sql,
                         Class elementType)
                  throws DataAccessException
```

入口参数：

sql 是查询数据库的 SQL 语句。

elementType 是查询结果的类型。

返回值：查询结果的 List 集合对象。

抛出异常：DataAccessException。

示例

获取用户表中所有用户的姓名，并输出结果。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String sql = "select name from tb_user";
    List list = jtm.queryForList(sql, String.class);
    for (Object object : list) {
        String name = (String) object;
        System.out.println("name=" + name);
    }
}
```

方法声明 3

该方法用于带参数的 SQL 多值查询，查询结果包含在方法返回的 List 集合中，集合中的每个元素都是 Map 集合对象，以数据表的字段名作为 Map 集合的 Key 名称。

```
public List queryForList(String sql,
                         Object[] args)
                  throws DataAccessException
```

入口参数：sql 是查询数据库的 SQL 语句。
返回值：包含查询结果的 List 集合对象。
抛出异常：DataAccessException。

示例

获取并输出用户表所有 27 岁男性的用户信息。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String sql = "select * from tb_user where age=? and sex=?";
    Object[] sqlArgs = {27, "男"};
    List list = jtm.queryForList(sql, sqlArgs);
    for (Object object : list) {
        Map map = (Map) object;
        System.out.print("\tmap=" + map.get("id"));
        System.out.print("\tmap=" + map.get("name"));
        System.out.print("\tmap=" + map.get("sex"));
        System.out.println("\tmap=" + map.get("age"));
    }
}
```

21.3.5 queryForLong 方法——long 类型单值查询方法

long 类型的单值查询方法。

方法声明 1

```
public long queryForLong(String sql)
                    throws DataAccessException
```

入口参数：sql 是查询数据库当前 SQL 语句。
抛出异常：DataAccessException。

示例

编写获取用户总数的方法。

```
public long getUserCount() {
    String sql="SELECT count(*) FROM tb_user";
    long count = getJdbcTemplate().queryForLong(sql);
    return count;
}
```

方法声明 2

支持 SQL 查询参数的方法。

```
public int queryForLong(String sql,
                        Object[] args)
                throws DataAccessException
```


PDF电子书基地 http://dayo1982.400gb.com

入口参数：
sql 是执行的 SQL 查询语句。
args 是 SQL 语句中对应的参数值。
返回值：查询的 long 类型数值。
抛出异常：DataAccessException。

示例

查询所有 27 岁男性用户数量。

```
public long getUserCount() {
    String sql="SELECT count(*) FROM tb_user WHERE sex=? and age=?";
    Object[] args={"男",27};
    long count = getJdbcTemplate().queryForLong(sql,args);
    return count;
}
```

21.3.6 queryForMap 方法——查询结果的 Map 集合

该方法用于获取单个查询结果的情况。

方法声明 1

```
public Map queryForMap(String sql)
                    throws DataAccessException
```

入口参数：sql 是执行的 SQL 查询语句。
返回值：查询结果的 Map 集合。
抛出异常：DataAccessException。

示例

获取 ID 编号是 1 的用户数据，并输出。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
                "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String sql = "select * from tb_user where id=1";
    Map map = jtm.queryForMap(sql);
    System.out.println(map);
}
```

方法声明 2

指定查询参数。

```
public Map queryForMap(String sql,
                       Object[] args)
                throws DataAccessException
```

入口参数：
sql 是执行的 SQL 查询语句。

args 是 SQL 语句中对应的参数值。

返回值：查询结果的 Map 集合。

抛出异常：DataAccessException。

示例

从数据库获取年龄为 27 岁的第一个男性用户，并输出该用户信息。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String sql = "select * from tb_user where age=? and sex=? limit 1";
    Object[] sqlArgs = {27, "男"};
    Map map = jtm.queryForMap(sql,sqlArgs);
    System.out.println(map);
}
```

21.3.7 queryForObject 方法——查询的结果以对象类型返回

该方法用于获取单值查询，查询的结果以对象类型返回。

方法声明

```
public Object queryForObject(String sql,
                             Class requiredType)
                      throws DataAccessException
```

入口参数：

sql 是执行的 SQL 查询语句。

requiredType 指定查询结果的类型。

返回值：指定查询类型的的结果对象。

抛出异常：DataAccessException。

示例

获取用户表中第一个用户的姓名。

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "cfg/XMLConfig.xml");
    Dao dao = (Dao) context.getBean("dao");
    JdbcTemplate jtm = dao.getJdbcTemplate();
    String sql = "select name from tb_user limit 1";
    String name = (String) jtm.queryForObject(sql, String.class);
    System.out.println(name);
}
```

21.3.8 update 方法——更新数据库中的数据

update 方法用于更新数据库中的数据。

PDF电子书基地 http://dayo1982.400gb.com

方法声明 1

执行更新数据的 SQL 语句。

```
public int update(String sql)
                throws DataAccessException
```

入口参数：sql 是执行的 SQL 查询语句。

返回值：影像到的数据记录数量。

抛出异常：DataAccessException。

示例

更改所有用户的年龄为 28 岁。

```
public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                        "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        JdbcTemplate jtm = dao.getJdbcTemplate();
        String sql = "update tb_user set age='28'";
        int update = jtm.update(sql);
        System.out.println("成功更新"+update+"位用户");
}
```

方法声明 2

执行带参数的更新数据的 SQL 语句。

```
public int update(String sql,
                        Object[] args)
                throws DataAccessException
```

入口参数：

sql 是执行的 SQL 查询语句。

args 是 SQL 语句中对应的参数值。

返回值：更新操作所影响的记录行数。

抛出异常：DataAccessException。

示例

将所有男性用户的年龄修改为 29 岁。

```
public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                        "cfg/XMLConfig.xml");
        Dao dao = (Dao) context.getBean("dao");
        JdbcTemplate jtm = dao.getJdbcTemplate();
        String sql = "update tb_user set age=? where sex=?";
        Object[] arg={29,"男"};
        int update = jtm.update(sql,arg);
        System.out.println("成功更新"+update+"位用户");
}
```

第 22 章　Spring Web 表单标签

Spring 的表单标签可以很容易地绑定 HTML 表单元素和控制器的表单对象，还可以绑定错误信息。

22.1　<form:checkbox>——复选框元素

该标签用于定义 HTML 表单的复选框元素。

语法

```
<form:checkbox path="path" value="value"/>
```

<form:form>标签属性表

属　性	类　型	描　述
path	String	绑定的表单对象的属性名
value	String	标签定义的值

示例

编写一个 JavaBean 来模拟一个控制器的表单对象，名称为 Fans，在该类中定义一个字符串数组的属性 fans，并提供访问该属性的方法。

```
public class Fans {
    private String[] fans;
    public String[] getFans() {
        return fans;
    }
    public void setFans(String[] fans) {
        this.fans = fans;
    }
}
```

在 JSP 页面使用<form:checkbox>标签定义复选框元素，该标签绑定之前定义的 JavaBean 的 fans 属性，与该属性匹配复选框的选择状态。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        com.Fans fanBean = new com.Fans();
        String[] fans = {"打球", "看电视", "体育"};
        fanBean.setFans(fans);
```


PDF电子书基地 http://dayo1982.400gb.com

```
            request.setAttribute("command", fanBean);
%>
爱好：<br>
<form:form>
    <form:checkbox path="fans" value="打球"/>打球
    <form:checkbox path="fans" value="看电视"/>看电视
    <form:checkbox path="fans" value="新闻"/>新闻
</form:form>
```

运行结果如图 22.1 所示。



图 22.1 运行结果

示例

编写一个 JavaBean 来模拟一个控制器的表单对象，名称为 Fans，在该类中定义一个字符串数组的属性 fans，并提供访问该属性的方法。

```
public class Fans {
    List fanList = new ArrayList();
    public List getFanList() {
        return fanList;
    }
    public void setFanList(List fanList) {
        this.fanList = fanList;
    }
}
```

在 JSP 页面使用<form:checkbox>标签定义复选框元素，该标签绑定之前定义的 JavaBean 的 fanList 属性，与该属性匹配复选框的选择状态。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        com.Fans fanBean = new com.Fans();
        List list=new ArrayList();
        list.add("周一");
        list.add("周二");
```

```
            list.add("周三");
            list.add("周四");
            list.add("周五");
            fanBean.setFanList(list);
            request.setAttribute("command", fanBean);
%>
工作日：<br>
<form:form>
    <form:checkbox path="fanList" value="周一"/>周一
    <form:checkbox path="fanList" value="周二"/>周二
    <form:checkbox path="fanList" value="周三"/>周三
    <form:checkbox path="fanList" value="周四"/>周四
    <form:checkbox path="fanList" value="周五"/>周五
    <form:checkbox path="fanList" value="周六"/>周六
    <form:checkbox path="fanList" value="周日"/>周日
</form:form>
```

运行结果如图 22.2 所示。

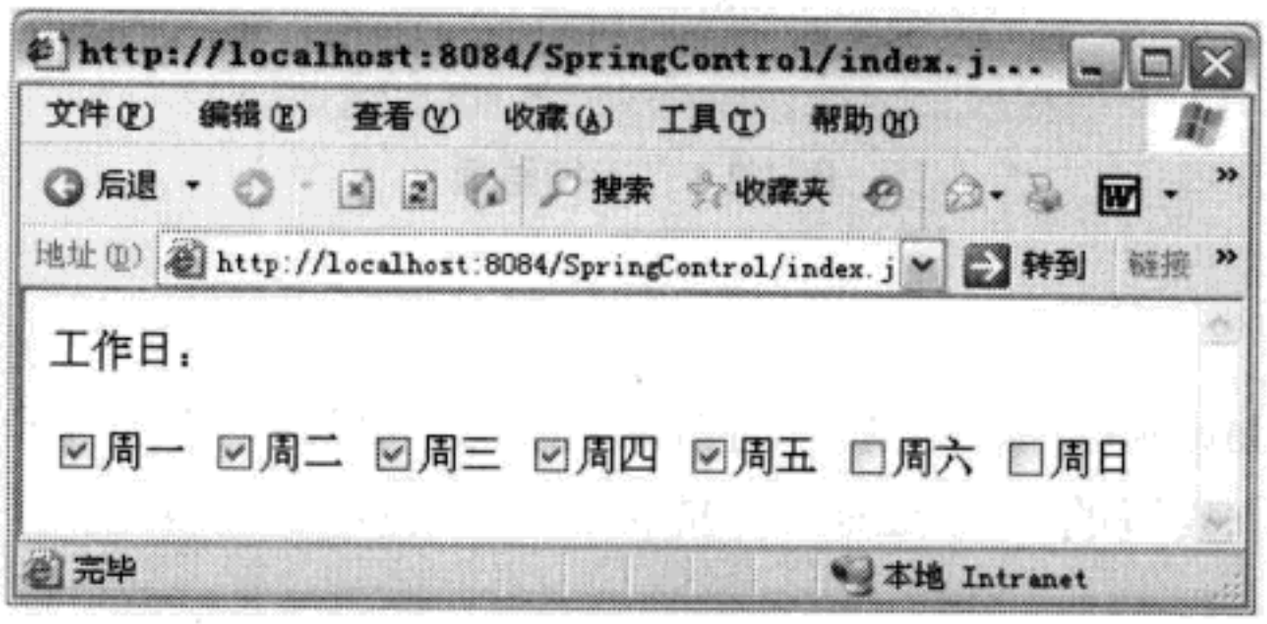


图 22.2 运行结果

22.2 <form:errors>——验证错误信息

该标签用于显示 Spring 控制器返回的验证错误信息。

语法

```
<form:errors path="pName" />
```

属性：path 是和 JavaBean 绑定的属性名称，该名称由 pName 指定。

示例

在表单中显示控制器返回的验证错误信息。

```
<form:form>
    用户名<form:input path="userName"/>
    <form:errors path="userName" />
    密码：<form:input path="password"/>
    <form:errors path="password"/>
</form:form>
```


PDF电子书基地 http://dayo1982.400gb.com

22.3 <form:form>——定义 HTML 表单

该标签用于定义 HTML 表单。除该标签以为的所有表单标签都是它的子标签，需要嵌套在该标签中使用。

语法 1

```
<form:form>
    …其他子标签
</form:form>
```

语法 2

```
<form:form action="/login.htm" commandName="userBean" method="POST">
    …其他子标签
</form:form>
```

<form:form>标签属性表

属　性	类　型	描　述
action	String	处理该表单的控制器路径
commandName	String	为绑定的控制器的数据模型定义一个名字，默认为“command”
method	String	指定表单提交数据所使用的方法

示例

定义用户登录表单。

```
<form:form action="/login.htm" commandName="userBean" method="POST">
    用户名<form:input path="userName"/>
    密码：<form:input path="password"/>
    <input type="submit" value="提交">
    <input type="reset" value="重置">
</form:form>
```

示例

如果表单是通过 Spring 的表单控制器转发的，那么就不需要指定 action 等参数，直接使用默认值就可以了。

```
<form:form">
    用户名<form:input path="userName"/>
    密码：<form:input path="password"/>
    <input type="submit" value="提交">
    <input type="reset" value="重置">
</form:form>
```

22.4 <form:hidden>——定义表单隐藏组件

该标签用于定义表单的隐藏组件。

语法

```
<form:hidden id="idName" path="beanName"/>
```

<form:hidden>标签属性表

属　性	类　型	描　述
id	String	这个隐藏组件的 id 名称
path	String	绑定的表单对象的属性名称

示例

在输入用户名的文本框组件下面定义一个隐藏组件。

```
<form:form action="/login.htm" commandName="userBean" method="POST">
    用户名<form:input path="userName"/>
    <form:hidden id="_name" path="userName"/>
    …省略其他代码
</form:form>
```

22.5 <form:input>——定义文本框

该标签用于定义表单的文本框输入组件，可以使用该组件接收用户输入。

语法

```
<form:input id="idName" path="beanProperty" maxlength="1"/>
```

<form:input>标签属性表

属　性	类　型	描　述
id	String	这个文本框输入组件的 id 名称
path	String	绑定的表单对象的属性名称
maxlength	int	文本框可以接收用户输入字符的数量

示例

编写一个简单的注册表单。

```
<form:form>
    用户名：<form:input maxlength="10" path="userName"/>
  …省略部分代码
    年龄：<form:input path="password"/>
</form:form>
```

22.6 <form:option>——定义下拉选择框

该标签是<form:select>标签的子标签，用于定义表单的下拉选择框或列表框的单个选项。

语法

```
<form:option label="label" value="value" />
```


PDF电子书基地 http://dayo1982.400gb.com

程序员的掌中宝

<form:option>标签属性表

属　性	类　型	描　述
label	String	选项的提示信息
value	String	选项的值

示例

用多个<form:option>标签定义表单的下拉选择框组件。

```
<form:form>
    证件类型:
    <form:select path="fanList" multiple="false">
        <form:option value="1" label="身份证"/>
        <form:option value="2" label="军官证"/>
        <form:option value="3" label="学生证"/>
    </form:select>
</form:form>
```

运行结果如图 22.3 所示。



图 22.3　运行结果

22.7 <form:options>——下拉框或列表框

该标签是<form:select>标签的子标签，用于定义表单的下拉选择框或列表框的多个选项。

语法

```
<form:options itemValue="itemValue" itemLabel="itemLabel" items="items" />
```

<form：options>标签属性表

属　性	类　型	描　述
items	String	选项的提示信息
itemValue	String	选项的值
itemLabel	String	选项的值

示例

在 Fans 类中定义 fanList 属性，模拟一个控制器的表单对象。

```
public class Fans {
    List fanList = new ArrayList();
    public List getFanList() {
        return fanList;
    }
```

```
        public void setFanList(List fanList) {
            this.fanList = fanList;
        }
    }
```

用<form:options>标签定义表单的下拉选择框组件。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        com.Fans fanBean = new com.Fans();
        List list = new ArrayList();
        list.add("周一");
        list.add("周二");
        list.add("周三");
        list.add("周四");
        list.add("周五");
        fanBean.setFanList(list);
        request.setAttribute("command", fanBean);
%>
<form:form>
    计划完成日期在本周的:
    <form:select path="fanList" multiple="false">
        <form:options items="${command.fanList}" />
    </form:select>
</form:form>
```

示例

在 Fans 类中定义 fanMap 属性，模拟一个控制器的表单对象。

```
public class Fans {
    Map fanMap;
    public Map getFanMap() {
        return fanMap;
    }
    public void setFanMap(Map fanMap) {
        this.fanMap = fanMap;
    }
}
```

用<form:options>标签定义表单的下拉选择框组件。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        com.Fans fanBean = new com.Fans();
        Map map=new TreeMap();
        map.put("1","周一");
```


PDF电子书基地 http://dayo1982.400gb.com

```
            map.put("3","周三");
            map.put("5","周五");
            map.put("7","周日");
            fanBean.setFanMap(map);
            request.setAttribute("command", fanBean);
%>
<form:form>
    下次招聘时间定在本周的:
    <form:select path="fanMap" multiple="false">
        <form:options items="${command.fanMap}" />
    </form:select>
</form:form>
```

22.8 <form:password>——定义密码框

该标签用于定义表单的密码框输入组件，可以使用该组件接收用户输入的密码。

语法

```
<form:input id="idName" path="beanProperty" maxlength="1"/>
```

<form:input>标签属性表

属　性	类　型	描　述
id	String	这个文本框输入组件的 id 名称
path	String	绑定的表单对象的属性名称
maxlength	int	文本框可以接收用户输入字符的数量

示例

模拟一个控制器的表单对象，使用 Spring Form 表单定义邮箱登录表单。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        class NewBean {
            private String email;
            private String password;
            public void setEmail(String email) {
                this.email = email;
            }
            public String getEmail() {
                return email;
            }
            public String getPassword() {
                return password;
            }
            public void setPassword(String pass) {
```

```
                    this.password = pass;
                }
            }
            NewBean fanBean = new NewBean();
            request.setAttribute("command", fanBean);
%>
<form:form>
    E-mail:<form:input path="email" size="25"/><br>
        密码：<form:password path="password" size="26"/><br>
    <input type="submit" value="登录">
    <input type="reset" value="重置">
</form:form>
```

运行结果如图 22.4 所示。

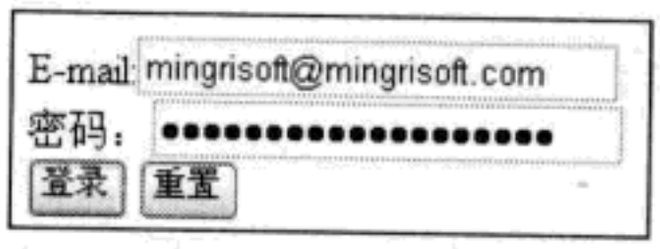


图 22.4 运行结果

22.9 <form:radiobutton>——定义单选按钮

该标签用于定义表单的单选按钮组件。单选按钮组件有两个或多个同名的标签组成。

语法

```
<form:radiobutton path=" path " value=" value " label=" label "/>
```

<form:input>标签属性表

属　性	类　型	描　述
label	String	标签的标题
path	String	绑定的表单对象的属性名称
value	int	单选按钮组件的标题

示例

模拟一个控制器的表单对象，使用 Spring Form 表单定义邮箱登录表单。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        class NewBean {
            private String sex;
            public String getSex() {
```


PDF电子书基地 http://dayo1982.400gb.com

```
            return sex;
        }
        public void setSex(String sex) {
            this.sex = sex;
        }
}
NewBean fanBean = new NewBean();
fanBean.setSex("女");
request.setAttribute("command", fanBean);
%>
<form:form>
    性别:<br>
    <form:radiobutton path="sex" value="男" label="男"/>
    <form:radiobutton path="sex" value="女" label="女"/><br>
    <input type="submit" value="登录">
    <input type="reset" value="重置">
</form:form>
```

22.10 <form:select>——下拉框或列表框

该标签用于定义表单的下拉选择框或者列表选择框。

语法

```
<form:options itemValue="itemValue" itemLabel="itemLabel" items="items" />
```

<form：options>标签属性表

属　性	类　型	描　述
items	String	选项的提示信息
itemValue	String	选项的值
itemLabel	String	选项的值

示例

在 NewBean 类中定义 fanList 属性，模拟一个控制器的表单对象。

```
public class NewBean {
    private List fanList;
    private String date;
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }
    public List getFanList() {
        return fanList;
    }
    public void setFanList(List fanList) {
```

```
            this.fanList = fanList;
        }
    }
```

用<form:select>标签定义表单的下拉选择框组件。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        com.NewBean fanBean = new com.NewBean();
        List list = new ArrayList();
        list.add("周一");
        list.add("周二");
        list.add("周三");
        list.add("周四");
        list.add("周五");
        fanBean.setFanList(list);
        request.setAttribute("command", fanBean);
%>
<form:form>
    计划完成日期在本周的:
    <form:select path="date" items="${command.fanList }"/>
</form:form>
```

运行结果如图 22.5 所示。

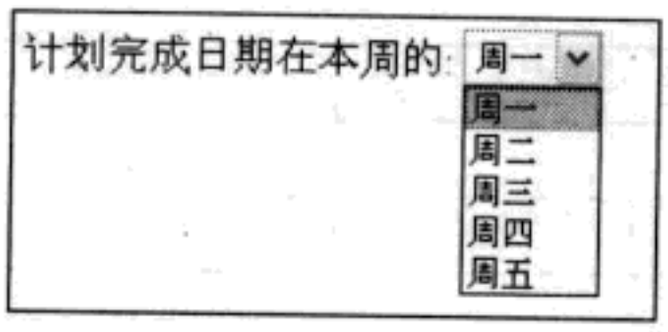


图 22.5 运行结果

22.11 <form:textarea>——定义文本域

该标签用于定义表单的文本域组件，它可以接收用户输入的大量文字，可以输入多行文本。

语法

```
<form:textarea path="path" cols=" cols " rows=" rows "/>
```

<form:options>标签属性表

属 性	类 型	描 述
path	String	与表单对象绑定的属性名称
cols	int	文本域组件每行字符数量
rows	int	文本域组件的行数

PDF电子书基地 http://dayo1982.400gb.com

示例

模拟一个控制器的表单对象，定义文本域组件。

```
<%@page contentType="text/html" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
        class NewBean {
            private String text="在这里输入文本";
            public void setText(String text) {
                this.text = text;
            }
            public String getText() {
                return text;
            }
        }
        NewBean fanBean = new NewBean();
        request.setAttribute("command", fanBean);
%>
<form:form>
    请输入你对本站的看法:<br>
        <form:textarea path="text" cols="20" rows="8"/>
</form:form>
```

附录 字母索引

<bean:cookie>	14.2 节
<bean:define>	14.1 节
<bean:header>	14.3 节
<bean:include>	14.1 节
<bean:message>	14.7 节
<bean:page>	14.5 节
<bean:parameter>	14.4 节
<bean:resource>	14.9 节
<bean:size>	14.6 节
<bean:struts>	14.11 节
<bean:write>	14.8 节
<c:catch>	9.1.4 节
<c:choose>	9.2.2 节
<c:forEach>	9.3.1 节
<c:forTokens>	9.3.2 节
<c:if>	9.2.1 节
<c:import>	9.4.1 节
<c:otherwise>	9.2.4 节
<c:out>	9.1.1 节
<c:param>	9.4.4 节
<c:redirect>	9.4.2 节
<c:remove>	9.1.3 节
<c:set>	9.1.2 节
<c:url>	9.4.3 节
<c:when>	9.2.3 节
<class>	19.2 节
<fmt:bundle>	10.8 节
<fmt:formatDate>	10.3 节

PDF电子书基地 http://dayo1982.400gb.com

续表

<fmt:formatNumber>	10.1 节
<fmt:message>	10.9 节
<fmt:param>	10.1 节
<fmt:parseDate>	10.4 节
<fmt:parseNumber>	10.2 节
<fmt:requestEncoding>	10.12 节
<fmt:setBundle>	10.7 节
<fmt:setlocale>	10.11 节
<fmt:setTimeZone>	10.5 节
<fmt:timeZone>	10.6 节
<form:checkbox>	22.1 节
<form:errors>	22.2 节
<form:form>	22.3 节
<form:hidden>	22.4 节
<form:input>	22.5 节
<form:option>	22.6 节
<form:options>	22.7 节
<form:password>	22.8 节
<form:radiobutton>	22.9 节
<form:select>	22.1 节
<form:textarea>	22.11 节
<generator>	19.4 节
<hibernate-mapping>	19.1 节
<html:html>	15.7 节
<html:base>	15.1 节
<html:checkbox>	15.2 节
<html:errors>	15.3 节
<html:file>	15.4 节
<html:form>	15.5 节
<html:hidden>	15.6 节
<html:image>	15.9 节
<html:img>	15.8 节
<html:link>	15.1 节

续表

<html:messages>	15.11 节
<html:multibox>	15.12 节
<html:option>	15.13 节
<html:optionCollection>	15.15 节
<html:options>	15.14 节
<html:password>	15.16 节
<html:radio>	15.17 节
<html:select>	15.19 节
<html:submit>	15.18 节
<html:text>	15.2 节
<html:textarea>	15.21 节
<id>	19.3 节
<jsp:fallback>	7.1 节
<jsp:forward>	7.2 节
<jsp:getProperty>	7.3 节
<jsp:include>	7.4 节
<jsp:param>	7.5 节
<jsp:params>	7.6 节
<jsp:plugin>	7.7 节
<jsp:setProperty>	7.8 节
<jsp:useBean>	7.9 节
<key>	19.5 节
<logic:empty>	16.9 节
<logic:equal>	16.1 节
<logic:forward>	16.16 节
<logic:greaterEqual>	16.3 节
<logic:greaterThan>	16.4 节
<logic:iterate>	16.15 节
<logic:lessEqual>	16.5 节
<logic:lessThan>	16.6 节
<logic:match>	16.7 节
<logic:messagesNotPresent>	16.14 节
<logic:messagesPresent>	16.13 节

PDF电子书基地 http://dayo1982.400gb.com

续表

<logic:notEmpty>	16.1 节
<logic:notEqual>	16.2 节
<logic:notMatch>	16.8 节
<logic:notPresent>	16.12 节
<logic:present>	16.11 节
<logic:redirect>	16.17 节
<many-to-many>	19.8 节
<many-to-one>	19.1 节
<one-to-many>	19.11 节
<one-to-one>	19.7 节
<property >	19.6 节
<set>	19.9 节
<sql:dateParam>	11.5 节
<sql:param>	11.4 节
<sql:query>	11.2 节
<sql:setDataSource>	11.1 节
<sql:transaction>	11.6 节
<sql:update>	11.3 节
<x:choose>	12.5 节
<x:forEach>	12.8 节
<x:if>	12.4 节
<x:otherwise>	12.7 节
<x:out>	12.2 节
<x:param>	12.1 节
<x:parse>	12.1 节
<x:set>	12.3 节
<x:transform>	12.9 节
<x:when>	12.6 节
absolute	5.5.2 节
add	4.1.1 节，4.3.1 节，17.6.1 节
addAll	4.1.2 节，4.3.2 节
addBatch	5.3.2 节
addClass	17.1.3 节

续表

addCookie	8.8.1 节
addDateHeader	8.8.2 节
addEntity	17.9.1 节
addErrors	13.1.1 节
addHeader	8.8.3 节
addIntHeader	8.8.4 节
addJoin	17.9.2 节
addMessages	13.1.2 节
addOrder	17.6.2 节
addResource	17.1.5 节
addScalar	17.9.3 节
afterLast	5.5.3 节
allEq	17.8.1 节
and	17.8.10 节
append	3.19.1 节
autoFlush	6.2.1 节
available	3.1.2 节，3.8.2 节，3.12.1 节
avg	17.7.6 节
batchUpdate	21.3.1 节
beforeFirst	5.5.4 节
begin	17.4.1 节
beginTransaction	17.3.1 节
bitCount	2.1.3 节，2.2.3 节
booleanValue	2.4.3 节
buffer	6.2.2 节
buildSessionFactory	17.1.4 节
byteValue	2.1.4 节，2.2.4 节，2.7.3 节，2.8.3 节
cache.provider_class	18.2 节
cache.region_prefix	18.21 节
cache.use_minimal_puts	18.22 节
cache.use_query_cache	18.18 节
cache.use_second_level_cache	18.19 节
cancel	5.3.3 节

PDF电子书基地 http://dayo1982.400gb.com

续表

cancelRowUpdates	5.5.5 节
canExecute	3.7.3 节
canRead	3.7.4 节
canWrite	3.7.5 节
charAt	1.2 节
charCount	2.6.3 节
charValue	2.6.4 节
clear	4.1.3 节，4.2.1 节，4.3.3 节，8.4.1 节，17.3.11 节
clearBatch	5.3.4 节
clearBuffer	8.4.2 节
clearParameters	5.4.1 节
clearWarnings	5.2.2 节，5.3.5 节，5.5.6 节
close	3.1.3 节，3.3.2 节，3.4.2 节，3.8.3 节，3.9.2 节，3.10.2 节，3.11.2 节，3.12.2 节，3.13.1 节，3.14.2 节，3.16.2 节，3.17.1 节，3.18.2 节，3.19.2 节，5.2.3 节，5.3.26 节，5.5.7 节，8.4.3 节，17.2.3 节，17.3.3 节
codePointAt	1.3 节，2.6.5 节
codePointBefore	1.4 节，2.6.6 节
codePointCount	1.5 节，2.6.7 节
commit	5.2.4 节，17.4.2 节
compare	2.7.4 节，2.8.4 节
compareTo	1.6 节，2.1.5 节，2.2.5 节，2.3.3 节，2.4.4 节，2.5.3 节，2.6.8 节，2.7.5 节，2.8.5 节，3.7.6 节
compareToIgnoreCase	1.7 节
concat	1.8 节
configure	17.1.2 节
connection.driver_class	18.2 节
connection.password	18.5 节
connection.url	18.3 节
connection.username	18.4 节
contains	1.9 节，4.1.4 节，4.3.4 节，17.3.12 节
containsAll	4.1.5 节，4.3.5 节
containsBean	20.5.1 节
containsHeader	8.8.5 节

续表

containsKey	4.2.2 节
containsValue	4.2.3 节
contentType	6.2.4 节
copyValueOf	1.1 节
count	17.7.11 节
createAlias	17.6.4 节
createBlob	5.2.5 节
createClob	5.2.6 节
createCriteria	17.3.15 节，17.6.3 节
createNClob	5.2.7 节
createNewFile	3.7.7 节
createQuery	17.3.13 节
createSQLQuery	17.3.14 节
createSQLXML	5.2.8 节
createStatement	5.2.9 节
createTempFile	3.7.8 节
decode	2.1.6 节，2.2.6 节，2.3.4 节，2.5.4 节
default_batch_fetch_size	18.13 节
default_catalog	18.1 节
default_entity_mode	18.14 节
default_schema	18.9 节
delete	3.7.9 节，17.3.7 节，21.2.2 节
deleteAll	21.2.3 节
deleteOnExit	3.7.10 节
deleteRow	5.5.8 节
delimiter	3.18.3 节
deregisterDriver	5.1.1 节
destroy	13.2.3 节
dialect	18.1 节
distinct	17.7.4 节
doGet	13.2.1 节
doPost	13.2.2 节
doubleToLongBits	2.7.7 节

PDF电子书基地 http://dayo1982.400gb.com

续表

doubleToRawLongBits	2.7.8 节
doubleValue	2.1.7 节，2.2.7 节，2.3.5 节，2.5.5 节，2.7.9 节，2.8.6 节
endsWith	1.11 节
eq	17.8.2 节
equals	1.12 节，2.1.8 节，2.2.8 节，2.3.6 节，2.4.5 节，2.5.6 节，2.6.9 节，2.7.10 节，2.8.7 节，3.7.11 节，4.1.6 节，4.2.4 节，4.3.6 节，8.5.1 节
equalsIgnoreCase	1.13 节
errorPage	6.2.5 节
evict	17.3.10 节
execute	5.3.6 节，5.4.2 节，13.1.3 节，13.3.1 节，13.4.1 节，13.5.1 节，13.6.1 节，13.8.1 节，21.2.4 节
executeBatch	5.3.7 节
executeFind	21.2.5 节
executeQuery	5.3.9 节，5.4.3 节
executeUpdate	5.3.8 节，5.4.4 节
exists	3.7.12 节
extends	6.2.6 节
fillInStackTrace	8.3.1 节
finalize	3.8.4 节，3.9.3 节
find	21.2.6 节
findColumn	5.5.9 节
findInLine	3.18.4 节
findWithinHorizon	3.18.5 节
first	5.5.10 节
floatToIntBits	2.8.8 节
floatToRawIntBits	2.8.9 节
floatValue	2.1.9 节，2.2.9 节，2.3.7 节，2.5.7 节，2.7.11 节，2.8.10 节
flush	3.2.2 节，3.4.3 节，3.6.1 节，3.11.3 节，3.13.2 节，3.14.3 节，3.19.3 节，8.4.4 节
format	1.14 节
format_sql	18.7 节
forward	8.6.1 节
ge	17.8.6 节

续表

generate_statistics	18.16 节
get	4.1.7 节，4.2.5 节，17.3.8 节，21.2.7 节
getAbsoluteFile	3.7.13 节
getAbsolutePath	3.7.14 节
getAsciiStream	5.5.11 节
getAttribute	8.6.3 节，8.7.1 节，8.9.1 节
getAttribute	8.1.1 节
getAttributeName	8.1.2 节
getAttributeNames	8.7.2 节，8.9.2 节
getAuthType	8.7.3 节
getAutoCommit	5.2.10 节
getBean	20.5.2 节
getBeanDefinitionCount	20.5.3 节
getBeanDefinitionNames	20.5.4 节
getBeanNamesForType	20.5.5 节
getBeansOfType	20.5.6 节
getBigDecimal	5.5.12 节
getBinaryStream	5.5.13 节
getBlob	5.5.14 节
getBoolean	2.4.6 节，5.5.15 节
getBufferSize	8.4.5 节
getByte	5.5.16 节
getBytes	1.15 节，5.5.17 节
getCanonicalFile	3.7.15 节
getCanonicalPath	3.7.16 节
getCatalog	5.2.11 节
getCause	8.3.2 节
getChannel	3.8.5 节，3.9.4 节，3.16.3 节
getCharacterEncoding	8.7.5 节，8.8.6 节
getCharacterStream	5.5.18 节
getChars	1.16 节
getClass	8.5.2 节
getClientInfo	5.2.12 节

PDF电子书基地 http://dayo1982.400gb.com

续表

getClob	5.5.19 节
getConcurrency	5.5.20 节
getConnection	5.1.4 节，5:3.10 节，21.1.2 节
getConnectionProperties	21.1.3 节
getContentLength	8.7.6 节
getContextPath	8.1.3 节，8.7.7 节
getCookies	8.7.4 节
getCreationTime	8.9.3 节
getCursorName	5.5.21 节
getDate	5.5.22 节
getDateHeader	8.7.8 节
getDouble	5.5.23 节
getDriver	5.1.2 节
getDriverClassName	21.1.4 节
getDrivers	5.1.3 节
getErrorData	8.6.2 节
getException	8.6.4 节
getFD	3.8.6 节，3.9.5 节，3.16.4 节
getFetchDirection	5.3.11 节，5.5.24 节
getFetchSize	5.3.12 节，5.5.25 节
getFilePointer	3.16.5 节
getFloat	5.5.26 节
getFreeSpace	3.7.17 节
getGeneratedKeys	5.3.13 节
getHeader	8.7.9 节
getHeaders	8.7.10 节
getHeadersNames	8.7.11 节
getHoldability	5.2.13 节，5.5.27 节
getId	8.9.4 节
getInitParameter	8.2.1 节
getInitParameterNames	8.2.2 节
getInputStream	8.7.12 节
getInstance	13.7.1 节

续表

getInt	5.5.28 节
getInteger	2.1.10 节
getIntHeader	8.7.13 节
getKeyMethodMap	13.6.2 节
getLastAccessedTime	8.9.5 节
getLocalizedMessage	8.3.4 节
getLoginTimeout	5.1.5 节
getLogWriter	5.1.6 节
getLong	2.2.10 节，5.5.29 节
getMajorVersion	8.1.4 节
getMaxFieldSize	5.3.14 节
getMaxInactiveInterval	8.9.6 节
getMaxRows	5.3.15 节，5.3.25 节
getMessage	8.3.3 节
getMetaData	5.2.14 节，5.4.5 节，5.5.30 节
getMethod	8.7.14 节
getMethodName	13.3.3 节，13.6.3 节
getMimiType	8.1.5 节
getMinorVersion	8.1.6 节
getModuleConfig	13.7.2 节
getMoreResults	5.3.16 节
getName	3.7.18 节
getNamedDispatcher	8.1.7 节
getNCharacterStream	5.5.31 节
getNClob	5.5.32 节
getNString	5.5.33 节
getNumericValue	2.6.10 节
getObject	5.5.34 节
getOutputStream	8.8.7 节
getPage	8.6.5 节
getParameter	8.7.15 节，13.3.2 节
getParameterMetaData	5.4.6 节
getParameterValues	8.7.16 节

PDF电子书基地 http://dayo1982.400gb.com

续表

getParent	3.7.19 节
getParentFile	3.7.20 节
getPassword	21.1.5 节
getPath	3.7.21 节
getPathInfo	8.7.17 节
getPathTranslated	8.7.18 节
getProtocol	8.7.19 节
getQueryString	8.7.20 节
getQueryTimeout	5.3.17 节
getRealPath	8.1.8 节
getRemaining	8.4.6 节
getRemoteAddr	8.7.24 节
getRemoteHost	8.7.25 节
getRemoteUser	8.7.21 节
getRequest	8.6.6 节
getRequestDispatcher	8.1.9 节
getRequestedSessionId	8.7.26 节
getRequestURI	8.7.22 节
getRequestURL	8.7.23 节
getResource	8.1.10 节
getResourceAsStream	8.1.11 节
getResourcePaths	8.1.12 节
getResources	13.1.5 节
getResponse	8.6.7 节
getResultSet	5.3.18 节
getResultSetConcurrency	5.3.19 节
getResultSetHoldability	5.3.20 节
getResultSetType	5.3.21 节
getRow	5.5.35 节
getServerName	8.7.28 节
getServerPath	8.7.29 节
getServerPort	8.7.30 节
getServletConfig	8.6.8 节

续表

getServletContext	8.2.3 节，8.6.9 节，8.9.7 节，20.5.10 节
getServletInfo	8.1.14 节
getServletName	8.2.4 节
getSession	8.6.10 节，8.7.27 节
getShort	5.5.36 节
getStackTrace	8.3.5 节
getStatement	5.5.37 节
getString	5.5.38 节
getTime	5.5.39 节
getTimestamp	5.5.40 节
getTotalSpace	3.7.22 节
getTransactionIsolation	5.2.15 节
getType	2.6.11 节，5.5.41 节，20.5.7 节
getTypeMap	5.2.16 节
getUpdateCount	5.3.22 节
getURL	5.5.42 节
getUrl	21.1.6 节
getUsableSpace	3.7.23 节
getUsername	21.1.7 节
getUserPrincipal	8.7.31 节
getWarnings	5.2.17 节，5.3.23 节，5.5.43 节
groupProperty	17.7.5 节
gt	17.8.5 节
handlePageException	8.6.11 节
hashCode	1.17 节，2.1.11 节，2.2.11 节，2.3.8 节，2.4.7 节，2.5.8 节，2.6.12 节，2.7.12 节，2.8.11 节，3.7.24 节，4.1.9 节，8.5.3 节
hasNext	3.18.6 节
hasNextBigDecimal	3.18.7 节
hasNextBigInteger	3.18.8 节
hasNextBoolean	3.18.9 节
hasNextByte	3.18.10 节
hasNextDouble	3.18.11 节
hasNextFloat	3.18.12 节

PDF电子书基地 http://dayo1982.400gb.com

续表

hasNextInt	3.18.13 节
hasNextLine	3.18.14 节
hasNextLong	3.18.15 节
hasNextShort	3.18.16 节
highestOneBit	2.1.12 节，2.2.12 节
id	17.7.2 节
import	6.2.7 节
in	17.8.9 节
include	8.6.12 节
indexOf	1.18 节，4.1.10 节
info	6.2.8 节
init	13.2.4 节
initCause	8.3.6 节
initialize	8.6.13 节
initInternal	13.2.5 节
initOther	13.2.6 节
insertRow	5.5.44 节
intBitsToFloat	2.8.12 节
intern	1.19 节
intValue	2.5.9 节，2.7.6 节，2.8.13 节
intValue	2.1.13 节，2.2.13 节，2.3.9 节
invalidate	8.9.8 节
ioException	3.18.17 节
isAbsolute	3.7.25 节
isAfterLast	5.5.45 节
isAutoFlush	8.4.7 节
isBeforeFirst	5.5.46 节
isClosed	5.2.18 节，5.3.27 节，5.5.47 节，17.2.2 节
isDefined	2.6.13 节
isDigit	2.6.14 节
isDirectory	3.7.26 节
isEmpty	1.2 节，4.1.12 节，4.2.6 节，4.3.7 节
isErrorPage	6.2.9 节

续表

isFile	3.7.27 节
isFirst	5.5.48 节
isHidden	3.7.28 节
isIEIgnored	6.2.10 节
isInfinite	2.7.13 节，2.8.14 节
.isLast	5.5.49 节
isLetter	2.6.15 节
isLowerCase	2.6.16 节
isNaN	2.7.14 节，2.8.15 节
isNew	8.9.9 节
isNotNull	17.8.13 节
isNull	17.8.14 节
isOpen	17.3.2 节
isPoolable	5.3.28 节
isReadOnly	5.2.19 节
isRequestedSessionIdFromCookie	8.7.32 节
isRequestedSessionIdFromURL	8.7.33 节
isRequestedSessionIdValid	8.7.34 节
isSingleton	20.5.8 节
isThreadSafe	6.2.11 节
isTokenValid	13.1.4 节
isTypeMatch	20.5.9 节
isUpperCase	2.6.17 节
isUserInRole	8.7.35 节
isValid	5.2.20 节
iterate	17.5.2 节
iterator	4.1.13 节，4.3.8 节
keySet	4.2.7 节
language	6.2.3 节
last	5.5.50 节
lastIndexOf	1.21 节，4.1.11 节
lastModified	3.7.29 节
le	17.8.4 节

PDF电子书基地 http://dayo1982.400gb.com

续表

length	1.22 节，3.7.30 节，3.16.6 节
like	17.8.8 节
list	3.7.31 节，17.5.1 节，17.6.5 节
listFiles	3.7.32 节
listIterator	4.1.14 节
listRoots	3.7.33 节
load	17.3.9 节
loadAll	21.2.8 节
locale	3.18.18 节
lock	17.3.16 节
log	8.1.13 节
longBitsToDouble	2.7.15 节
longValue	2,1.14 节，2.2.14 节，2.3.10 节，2.5.10 节，2.7.16 节，2.8.16 节
lowestOneBit	2.1.15 节，2.2.15 节
lt	17.8.3 节
mark	3.1.4 节，3.3.3 节，3.12.3 节，3.17.2 节
markSupported	3.1.5 节，3.3.4 节，3.12.4 节，3.17.3 节
match	3.18.19 节
matches	1.23 节
max	17.7.8 节
max_fetch_depth	18.12 节
min	17.7.9 节
mkdir	3.7.34 节
mkdirs	3.7.35 节
moveToCurrentRow	5.5.51 节
moveToInsertRow	5.5.52 节
nativeSQL	5.2.21 节
newLine	3.4.4 节，8.4.8 节
next	3.18.20 节，5.5.53 节
nextBigDecimal	3.18.21 节
nextBigInteger	3.18.22 节
nextBoolean	3.18.23 节

续表

nextByte	3.18.24 节
nextDouble	3.18.25 节
nextFloat	3.18.26 节
nextInt	3.18.27 节
nextLine	3.18.28 节
nextLong	3.18.29 节
nextShort	3.18.30 节
not	17.8.12 节
offsetByCodePoints	1.24 节
openSession	17.2.1 节
or	17.8.11 节
order_updates	18.15 节
pageEncoding	6.2.12 节
parseBoolean	2.4.8 节
parseByte	2.5.11 节
parseDouble	2.7.17 节
parseFloat	2.8.17 节
parseInt	2.1.16 节
parseLong	2.2.16 节
parseShort	2.3.11 节
prepareCall	5.2.22 节
prepareStatement	5.2.23 节
previous	5.5.54 节
print	8.4.9 节
println	5.1.7 节，8.4.10 节
printStackTrace	8.3.7 节
process	13.2.7 节
projectionList	17.7.1 节
property	17.7.3 节
put	4.2.8 节
putAll	4.2.9 节
query	21.3.2 节
queryForInt	21.3.3 节

PDF电子书基地 http://dayo1982.400gb.com

续表

queryForList	21.3.4 节
queryForLong	21.3.5 节
queryForMap	21.3.6 节
queryForObject	21.3.7 节
radix	3.18.31 节
read	3.1.6 节，3.3.5 节，3.5.1 节，3.8.7 节，3.10.3 节，3.12.5 节，3.16.7 节，3.17.4 节
readBoolean	3.5.3 节，3.16.8 节
readByte	3.5.2 节，3.16.9 节
readchar	3.5.4 节，3.15.5 节
readChar	3.16.11 节
readDouble	3.5.5 节，3.16.12 节
readFloat	3.5.6 节，3.15.4 节，3.16.13 节
readFully	3.5.7 节，3.16.14 节
readInt	3.5.8 节，3.15.3 节，3.16.10 节
readLine	3.3.6 节，3.16.15 节
readLong	3.5.9 节，3.16.16 节
readObject	3.15.2 节
readShort	3.5.10 节，3.16.17 节
readUnsignedByte	3.5.11 节，3.16.18 节
readUnsignedShort	3.5.12 节，3.16.19 节
readUTF	3.5.13 节，3.16.20 节
ready	3.3.7 节，3.17.5 节
regionMatches	1.25 节
registerDriver	5.1.8 节
relative	5.5.55 节
release	8.6.14 节
releaseSavepoint	5.2.31 节
remove	4.1.15 节，4.2.10 节，4.3.9 节
removeAll	4.1.16 节，4.3.10 节
removeAttribute	8.6.15 节，8.7.36 节，8.9.12 节
removeAttribute	8.1.15 节
renameTo	3.7.36 节

续表

replace	1.26 节
replaceAll	1.27 节
replaceFirst	1.28 节
reset	3.1.7 节，3.3.8 节，3.12.6 节，3.17.6 节，3.18.32 节
resetToken	13.1.6 节
retainAll	4.1.17 节，4.3.11 节
reverse	2.1.17 节，2.2.17 节
reverseBytes	2.1.18 节，2.2.18 节，2.3.12 节
rollback	5.2.24 节，17.4.3 节
rowCount	17.7.10 节
save	17.3.4 节，21.2.9 节
saveErrors	13.1.8 节
saveMessages	13.1.9 节
saveOrUpdate	17.3.6 节，21.2.10 节
saveOrUpdateAll	21.2.11 节
saveToken	13.1.7 节
seek	3.16.22 节
sendError	8.8.8 节
sendRedirect	8.8.9 节
session	6.2.13 节
session_factory_name	18.11 节
set	4.1.8 节
setAsciiStream	5.4.7 节
setAttibute	8.9.10 节
setAttribute	8.6.16 节，8.7.37 节
setAutoCommit	5.2.25 节
setBigDecimal	5.4.8 节
setBinaryStream	5.4.9 节
setBlob	5.4.10 节
setBoolean	5.4.11 节，17.5.11 节
setByte	5.4.12 节
setBytes	5.4.13 节
setCatalog	5.2.26 节

PDF电子书基地 http://dayo1982.400gb.com

续表

setCharacterStream	5.4.14 节
setClientInfo	5.2.27 节
setClob	5.4.15 节
setConnectionProperties	21.1.8 节
setContentType	8.8.10 节
setCursorName	5.3.29 节
setDate	5.4.16 节，17.5.8 节
setDateHeader	8.8.11 节
setDouble	5.4.17 节，17.5.12 节
setDriverClassName	21.1.9 节
setEntity	17.5.6 节
setEscapeProcessing	5.3.30 节
setExecutable	3.7.37 节
setFetchDirection	5.3.31 节，5.5.56 节
setFetchSize	5.3.32 节，5.5.57 节
setFirsResult	17.5.5 节，17.6.7 节
setFloat	5.4.18 节，17.5.13 节
setHeader	8.8.12 节
setHoldability	5.2.28 节
setInt	5.4.19 节
setInteger	17.5.10 节
setLastModified	3.7.38 节
setLength	3.16.21 节
setLoginTimeout	5.1.9 节
setLogWriter	5.1.10 节
setLong	5.4.20 节
setMaxFieldSize	5.3.33 节
setMaxInactiveInterval	8.9.11 节
setMaxResults	17.5.4 节，17.6.6 节
setMaxRows	5.3.24 节，5.3.34 节
setNCharacterStream	5.4.21 节
setNClob	5.4.22 节
setNString	5.4.23 节

续表

setParameter	17.5.7 节
setPassword	21.1.10 节
setPoolable	5.3.35 节
setProjection	17.6.9 节
setProperty	17.1.6 节
setQueryTimeout	5.3.36 节
setReadable	3.7.39 节
setReadOnly	3.7.40 节，5.2.29 节
setSavepoint	5.2.30 节
setShort	5.4.24 节
setStackTrace	8.3.8 节
setStatus	8.8.13 节
setString	5.4.25 节，17.5.9 节
setTime	5.4.26 节
setTimestamp	5.4.27 节
setTransactionIsolation	5.2.32 节
setURL	5.4.30 节
setUrl	21.1.11 节
setUsername	21.1.12 节
setWritable	3.7.41 节
shortValue	2.1.19 节，2.2.19 节，2.3.13 节，2.5.12 节，2.7.18 节，2.8.18 节
show_sql	18.6 节
signum	2.1.20 节，2.2.20 节
size	3.6.2 节，4.1.18 节，4.2.11 节，4.3.12 节
skip	3.1.8 节，3.3.9 节，3.8.8 节，3.12.7 节，3.17.7 节，3.18.33 节
skipBytes	3.5.14 节，3.16.23 节
split	1.29 节
sqlRestriction	17.8.15 节
startsWith	1.3 节
subList	4.1.19 节
subSequence	1.31 节
substring	1.32 节

PDF电子书基地 http://dayo1982.400gb.com

续表

sum	17.7.7 节
toArray	4.1.20 节，4.3.13 节
toBinaryString	2.1.21 节，2.2.21 节
toCharArray	1.33 节
toHexString	2.1.22 节，2.2.22 节，2.7.19 节，2.8.19 节
toLowerCase	1.34 节，2.6.18 节
toOctalString	2.1.23 节，2.2.23 节
toString	2.1.24 节，2.2.24 节，2.3.14 节，2.4.9 节，2.5.13 节，2.7.20 节，2.8.20 节，3.7.42 节，3.18.34 节，8.3.9 节
toUpperCase	1.35 节，2.6.19 节
toURI	3.7.43 节
transaction. factory_class	18.23 节
trim	1.36 节
uniqueResult	17.5.3 节，17.6.8 节
update	17.3.5 节，21.2.12 节，21.3.8 节
updateAsciiStream	5.5.58 节
updateBigDecimal	5.5.59 节
updateBinaryStream	5.5.60 节
updateBlob	5.5.61 节
updateBoolean	5.5.62 节
updateByte	5.5.63 节
updateBytes	5.5.64 节
updateCharacterStream	5.5.65 节
updateDate	5.5.66 节
updateDouble	5.5.67 节
updateFloat	5.5.68 节
updateInt	5.5.69 节
updateLong	5.5.70 节
updateNString	5.5.71 节
updateNull	5.5.72 节
updateRow	5.5.73 节
updateShort	5.5.74 节
updateString	5.5.75 节

续表

updateTime	5.5.76 节
updateTimestamp	5.5.77 节
use_identifer_rollback	18.17 节
use_sql_comments	18.8 节
useDelimiter	3.18.35 节
useLocale	3.18.36 节
useRadix	3.18.37 节
valueOf	1.37 节，2.1.25 节，2.2.25 节，2.3.15 节， 2.4.10 节，2.5.14 节，2.7.21 节，2.8.21 节
values	4.2.12 节
wasCommitted	17.4.4 节
wasRolledBack	17.4.5 节
write	3.2.3 节，3.4.5 节，3.6.3 节，3.9.6 节，3.11.4 节， 3.13.3 节，3.14.6 节，3.16.24 节，3.19.4 节
writeBoolean	3.6.4 节，3.16.25 节
writeByte	3.6.5 节，3.16.26 节
writeBytes	3.6.6 节，3.14.5 节，3.16.27 节
writeChar	3.6.7 节，3.14.7 节
writeChars	3.14.8 节，3.16.28 节
writechars	3.6.8 节
writeDouble	3.6.9 节，3.14.9 节，3.16.29 节
writeFields	3.14.10 节
writeFloat	3.6.10 节，3.14.11 节，3.16.30 节
writeInt	3.14.12 节，3.16.31 节
writeLong	3.6.11 节，3.14.13 节，3.16.32 节
writeObject	3.14.4 节
writeShort	3.6.12 节，3.14.14 节，3.16.33 节
writeUTF	3.6.13 节，3.14.15 节，3.16.34 节

PDF电子书基地 http://dayo1982.400gb.com

本书主要内容：

字符串	JSP动作与指令	Hibernate常用API
包装类	JSP内置对象	Hibernate配置及映射
输入/输出	JSTL标签库	Spring容器
集合类	Struts控制器组件	Spring持久化
数据库编程	Struts核心标签库	Spring Web应用标签库

全程跟踪服务

周一至周五8:00~17:00全程服务，为您解答编程问题

QQ、TQ、电话、论坛，四位立体式服务，让您编程畅通无阻

编程学习社区(bbs.mrbccd.com)，学、练、用，让您无所不能

服务热线：400-675-1066 0431-84978981 0431-84978982

QQ：100310063 1026560213

TQ：登录www.mingribook.com直接与客服交流

图书论坛：bbs.mingribook.com

ISBN 978-7-115-21487-4

9 787115 214874 >

ISBN 978-7-115-21487-4

定价：59.00 元

封面设计：黄 凯

分类建议：计算机/程序设计/Java

人民邮电出版社网址：www.ptpress.com.cn